1 Red Flag Linux 基础

Linux 是一种多用户、多任务的类 Unix 风格的操作系统，以高效和灵活著称。其独特之处在于它不受任何商品化软件的版权制约，全世界都能免费、自由使用。事实上，Linux 也是一种通用的操作系统，本章将带领大家一睹 Linux 操作系统的风采，你会发现在 Windows 上进行的操作在 Linux 中也可以进行。

1.1 Linux 简介

1.1.1 什么是 Linux

Linux 最早是由芬兰赫尔辛基大学计算机系大学生 Linus Torvalds 开发设计的。Linus Torvalds 最初只是为自己的操作系统课程编写了一个进程切换器，然后为了方便自己上网编写了终端仿真程序，再后来为了从网上下载文件编写了硬盘驱动程序和文件系统，此时他发现自己已经实现了一个几乎完整的操作系统内核，出于对内核的信心和美好的奉献精神和自由精神，他希望这个内核能够免费扩散使用。1991 年 8 月他完成了 Linux 0.01 版本，后来，在热心支持者的帮助下逐渐开发和推出了 0.10 版本、0.11 版本和 0.12 版本，1994 年 3 月，出现了带宣言意味的 Linux 1.0 版本。

Linux 自诞生以来发展迅速。1992 年 1 月，全世界大约只有 100 人使用 Linux，但由于它通过 Internet 发布，任何人均可获得其基本文件，并可对其进行修补和上传，在众多热心者的努力下，Linux 逐渐成为一个功能完善、稳定可靠且被广泛使用的操作系统。

Linux 不仅为广大用户提供了在 PC 机上学习和使用 Unix 操作系统的机会，同时 Linux 强大完善的功能和稳定性也使其拥有了许多一流的企业用户和团体用户，其中包括 NASA、迪斯尼、通用电器、波音、NASDAQ 以及一些一流的大学机构等，许多大型公司如：IBM、Dell、HP、Oracle、AMD 等也均加入了 Linux 的开发队伍，为 Linux 的发展贡献着力量。

在我国，Linux 起步较晚，但是随着 Linux 在各个行业的广泛成功应用，企业对 Linux 人才的需求与日俱增，越来越多的人感觉到学习 Linux 的迫切性，加入到学习、应用 Linux 的行列。

1.1.2 Linux操作系统的组成

Linux系统一般有四个主要部分：内核、Shell、文件系统和应用程序。

(1) Linux内核：内核是系统的“心脏”，是运行程序和管理磁盘、打印机等硬件设备的核心程序。

(2) Linux Shell：Shell是系统的用户界面，提供了用户与内核进行交互操作的一种接口。它接受用户输入的命令，并对其进行解释，最后送入内核去执行，实际上就是一个命令解释器。人们也可以使用Shell编程语言编写Shell程序，这些Shell程序与用其他程序设计语言编写的应用程序具有相同的效果。

(3) Linux文件系统：文件系统是文件存放在磁盘等存储设备上的组织方法。Linux的文件系统呈树形结构，同时它也能支持目前流行的文件系统，如：EXT2、EXT3、FAT、VFAT、NFS、SMB等。

(4) Linux应用程序：同Windows操作系统一样，标准的Linux也提供了一套满足人们上网、办公等需求的程序集即应用程序，包括文本编辑器、X Window、办公套件、Internet工具、数据库等。

Linux内核、Shell和文件系统一起形成了基本的操作系统结构，可使用户运行程序、管理文件并使用系统。

1.1.3 Linux的内核版本与发行版本

Linux不断发展推陈出新，跟Windows系列一样拥有不同的版本。但是Linux的版本号分为两部分，内核(Kernel)版本与发行套件(Distribution)版本。

1. Linux的内核版本

内核版本是在Linux的创始人Linus领导下的开发小组开发出的系统内核的版本号。由三个数字组成：r. x. y，其中，r表示目前发布的Kernel主版本；x表示开发中的版本；y表示错误修补的次数。一般来讲，x为偶数的版本表明该版本是一个可以使用的稳定版本，如2.4.4；x为奇数的版本表明该版本是一个测试版本，其中加入了一些新的内容，不一定稳定，如2.1.111。

Red Flag Linux桌面7.0使用的内核版本是2.6.25。表明Red Flag Linux桌面7.0使用的是一个比较稳定的版本，修补了25次。

2. Linux的发行版本

发行版本是一些组织或厂商将Linux系统内核与应用软件和文档包装起来，并提供一些安装界面和系统设定管理工具的一个软件包的集合。相对于内

核版本，发行套件的版本号随发布者的不同而不同，与系统内核的版本号相对独立。如 Red Flag Linux 桌面 7.0 指 Linux 的发行版本号，而其使用的内核版本是 2.6.25。

Linux 是自由软件，任何组织、厂商和个人都可以按照自己的要求进行发布，目前已经有了 300 余种发行版本，而且数目还在不断增加。Red Hat Linux、Fedora Core Linux、Debian Linux、Turbo Linux、Slackware Linux、Open Linux、SUSE Linux 和 Red Flag Linux(红旗 Linux，中国发布)等都是流行的 Linux 发行版本。

1.1.4 Linux 应用

Linux 操作系统可以开发支持几乎任何一种应用程序。目前，Linux 应用程序主要有以下几种：

- 文本和文字处理程序。除了一些商业化文字处理软件外，Linux 还提供了功能强大的文字处理软件如 vi 等。
- 办公软件。为了方便用户处理工作文档，Linux 中有一些类似微软 Office 办公系列软件的办公套件，如 OpenOffice.org 等包括文字处理、电子表格和演示文稿等。
- X Window。X Window 是 Unix 的图形化用户界面，可运行在 Linux 等类 Unix 操作系统上。在 X Window 上运行的大量应用程序使 Linux 成为易于使用的操作系统。
- 编程语言。Linux 可运行多种编程工具，编写并执行多种编程语言和脚本语言。Linux 的廉价性、灵活性、安全性及稳定性，已开始吸引越来越多的编程人员将自己的编程环境建立在 Linux 操作系统之上。
- Internet 工具。Linux 提供并支持各种 Internet 软件，如浏览器、邮件管理器、建立 Internet 服务所需的软件以及对建立网络连接进行支持的软件等。事实上，许多大型的网络服务商的服务器上运行的操作系统就是 Linux。
- 数据库。Linux 上不仅可以运行免费的 MySQL 和 Postgre 之类强大的免费数据库，随着 Linux 的不断普及，一些大型的数据库公司如 Oracle、Sybase 和 Informix 都提供了适用于 Unix、Linux 的关系型数据库产品。
- 娱乐。Linux 提供了大量的娱乐软件，包括音频播放器、视频播放器、录音机等，甚至还有十几款有趣的游戏。

经过短短十多年的发展，Linux 已成为最流行的操作系统之一，广泛应用于教育、科研、军事、企业以及个人计算机领域。良好的移植性、硬件兼容性、稳定高效，使它可以方便并可靠地部署在超级计算机、工作站、数据存储、网络服务器之上。2010 年 11 月的数据显示，在超级计算机 TOP500 中，有 459 台运行着 Linux，占到 91.8%，像《泰坦尼克号》、《我是传奇》、《指环王》、《星球大战》、《哈利波特》、《怪物

史莱克》、《2012》、《阿凡达》等特效制作都是在基于 Linux 的集群上完成；同样优秀的 Web 服务器 Apache 以完美的 LAMP(Linux＋Apache＋MySQL＋PHP)架构正在大行其道；各种大型的数据库，解决方案纷纷应用于 Linux 之上；新型的技术如虚拟化、云计算广泛基于 Linux 架构之上。随着 X Window 的加入，桌面环境发展和应用软件的极大丰富，Linux 在图形界面易用性上也取得了长足的进步；在嵌入式领域，Linux 更是广泛应用于战斗机、手持设备(如手机、PDA)、上网本、消费电子产品、家电智能操控、广告牌、汽车电子、网络设备、导航设备、工业机器人等。

1.2 Red Flag Linux 的产生与发展

Red Flag Linux 是北京中科红旗软件技术有限公司(红旗软件)开发并发行的 Linux 操作系统的发行版。

1999 年 8 月 10 日，红旗 Linux 诞生。同年 10 月 20 日，服务器版 1.0 正式上市。

2000 年 8 月 4 日，红旗 Linux 桌面版 2.0 发布；TCL 同时发布预装红旗 Linux 桌面版的个人电脑。同年 9 月，红旗发布了基于 IBM S/390 主机的红旗 Linux 操作系统，教育部考试中心指定红旗 Linux 为国家 NIT 体系的 Linux 模块的考试模板。同年 10 月，红旗 Linux 发布嵌入式解决方案，包括机顶盒、PDA、瘦客户机等。

2001 年 12 月，红旗企业级服务器 3 系列推出，开始进入企业市场。

2002 年 3 月，红旗 Linux 桌面版 3.0 推出。

2003 年 7 月，红旗 Linux 推出红旗 Linux 4 系列。

2004 年 6 月，在红旗软件的倡导下，Asianux 1.0 发布。10 月，红旗软件推出红旗 Linux 桌面版 4.1。

2005 年 8 月，红旗软件召开 Red Flag World 大会，并同时发布服务器 5 系列产品。

2006 年 3 月，发布红旗 Linux 桌面版 5.0(代号 Apatite)。

2007 年，发布内核为 kernel-2.6.22.6 的红旗 Linux 桌面版 6.0。

2009 年 6 月，发布红旗 Linux 桌面版 7.0。

1.3 Red Flag Linux 桌面 7.0 的安装

安装 Red Flag Linux 桌面 7.0 之前，需要做安装前期的准备工作，主要包括备份数据、搜集系统资料、准备硬盘分区。

在安装系统之前，最好将硬盘上的重要数据备份到软盘、光盘、U 盘或移动硬

盘上。通常要做的备份内容包括系统分区表、个人生成的重要文件与数据等。

安装系统时确保硬件环境符合以下需求：

(1) 4GB以上的硬盘空间；

(2) 不小于512MB的内存，内存1GB以上会有更好的使用感受。

一块硬盘可以被分为多个分区，分区之间是相互独立的。分区有三种类型：主分区(primary-partition)、扩展分区(extended-partition)和逻辑分区(logical-partition)。一个硬盘最多可以有四个主分区，如果想在一块硬盘上划分四个以上的分区，就要创建扩展分区，然后再扩展分区上划分出逻辑分区。Red Flag Linux既可以安装在主分区上，也可以安装在逻辑分区上。

注意，在安装的过程中，程序会自动对分区进行检查，所以只有确保系统有足够的硬盘空间(不包括swap分区在内有4GB以上)才能保证Red Flag Linux的顺利安装。

由于Red Flag Linux桌面7.0支持LiveCD，即可以从光盘直接进入系统，然后再进行安装。因此安装过程要比以前的系统安装简单、轻松得多，进入系统方法有如下两种：

(1) 从光盘启动即可直接使用系统；

(2) 制作LiveUSB系统(即通过U盘启动，Linux或Windows均可创建)。

至于从光盘启动使用系统，用户只需要到红旗的官方网站：http://www.redflag-linux.com/下载Red Flag Linux桌面7.0的镜像文件，然后将其刻成光盘。只需将刻好的光盘放入光驱，在启动计算机时设置成从光驱启动，即可进入Red Flag Linux桌面7.0系统进行体验了。还有一种更加直接的办法是先进入计算机原先的操作系统(Windows与Linux均可)，将刻好的光盘放入光驱，点击光驱的图标就可以进入Red Flag Linux桌面7.0系统。

下面我们介绍使用UNetbootin制作LiveUSB，然后从U盘启动进入Red Flag Linux桌面7.0系统的方法。

这种方法不用刻录光盘，对硬盘数据毫无影响。只要有一个1GB或以上的U盘或移动硬盘，而且U盘或移动硬盘和电脑都支持从USB设备启动，即可轻易体验Red Flag Linux桌面7.0的华丽。并且整个制作安装过程只需几分钟。

也可以用这个LiveUSB来把Red Flag Linux正式安装到硬盘里。

制作LiveUSB首先需要用到UNetbootin这个软件，UNetbootin (Universal Netboot Installer)为一种跨平台工具软件，有Windows版和Linux版，就算现在使用的是Windows系统，也可以轻松体验。可以用来建立LiveUSB系统，也可以加载各种系统工具，或安装各种Linux操作系统(Linux套件)和其他操作系统，不需使用安装光碟(自动通过网络下载)。

UNetbootin可以在Windows或Linux里使用，而且制作的速度快，支持的

Linux 发行版也很多。

LiveUSB 制作步骤如下：

(1) 下载 Red Flag Linux 桌面 7.0 的 iso 文件。

(2) 下载 UNetbootin。Linux 用户下载"for Linux"的版本即可。

(3) 将 U 盘插到 USB 接口。注意：一定要先插 U 盘再运行 UNetbootin!

(4) 运行 UNetbootin。

Windows 版是免安装的绿色软件，双击下载来的文件即可运行。

Linux 版的文件也是免安装的，但要先赋予它可执行的权限，Linux 用户可用右键单击下载来的文件，选择"属性"，点击"权限"标签，选中"允许以程序执行文件"，点"关闭"按钮。然后双击文件，即可运行程序。

(5) 选择"磁盘镜像"(Diskimage)和下载好的 iso 文件。

在 Windows 下，不要选择"显示所有驱动器"(Show All Drives)。"类型"(Type)选择"USB 设备"(USB Drive)，"驱动器"选择你的 U 盘。注意：千万不要选错驱动器，如果错选为硬盘的话，会导致硬盘文件丢失!

Linux 下，如果你不确定哪个是你的 U 盘的话，可以打开菜单"系统 → 系统管理 → 分区编辑器"确认一下，根据驱动器容量即可分辨。

(6) 确认选择正确后，点"确认"(OK)开始安装。

(7) 只需几分钟即可安装完毕。

(8) 重启电脑，并设置 USB 设备为第一启动设备，即可启动 U 盘里的操作系统。

这样创建的 LiveUSB 类似于 LiveCD，其优缺点如下。

缺点：

(1) 不能保存个人文件或系统设置(可以把文件保存到其他驱动器，如另一个 U 盘或移动硬盘，或电脑里的硬盘)；

(2) 需要电脑支持 USB 设备启动。

优点：

(1) 制作方便；

(2) 运行速度比光盘快。

移动硬盘的做法跟 U 盘一样。

UNetbootin 也支持 Fedora、openSUSE 等 Linux 发行版，详细列表可以到官方主页查阅。

当用户选择从光盘或 U 盘进入 Red Flag Linux 系统后，系统首先会进入到如下界面(图 1.1)。

当用户等待几秒钟后，系统会提示进行语言、键盘及时区的设置界面(图 1.2)。

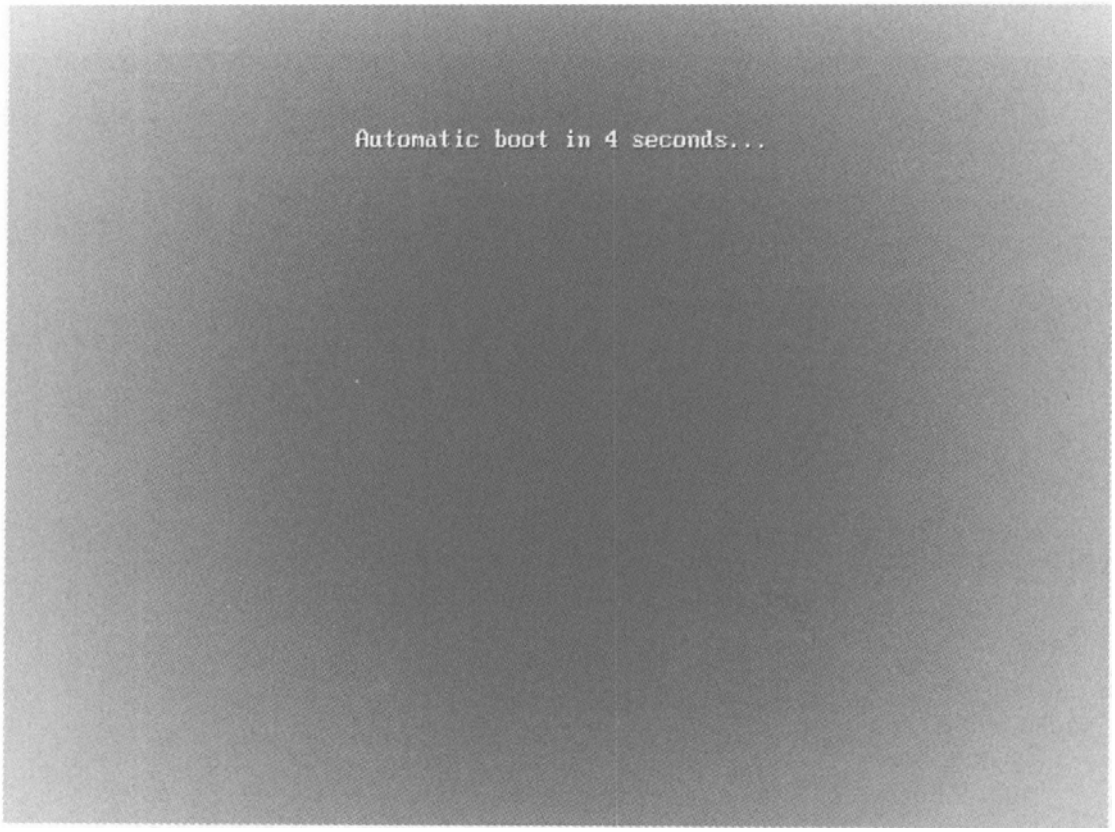


图 1.1　进入系统前的等待界面

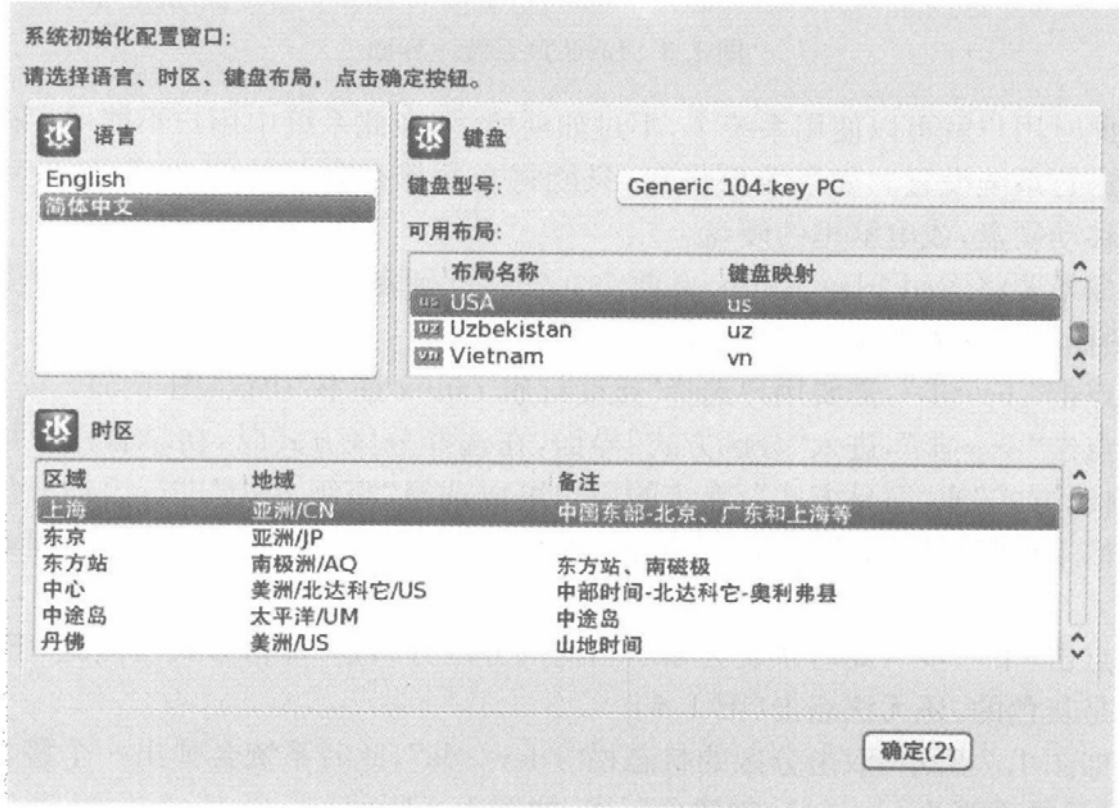


图 1.2　语言、键盘及时区设置界面

当设置完成后，便可进入如图 1.3 所示的 LiveCD 系统主界面。

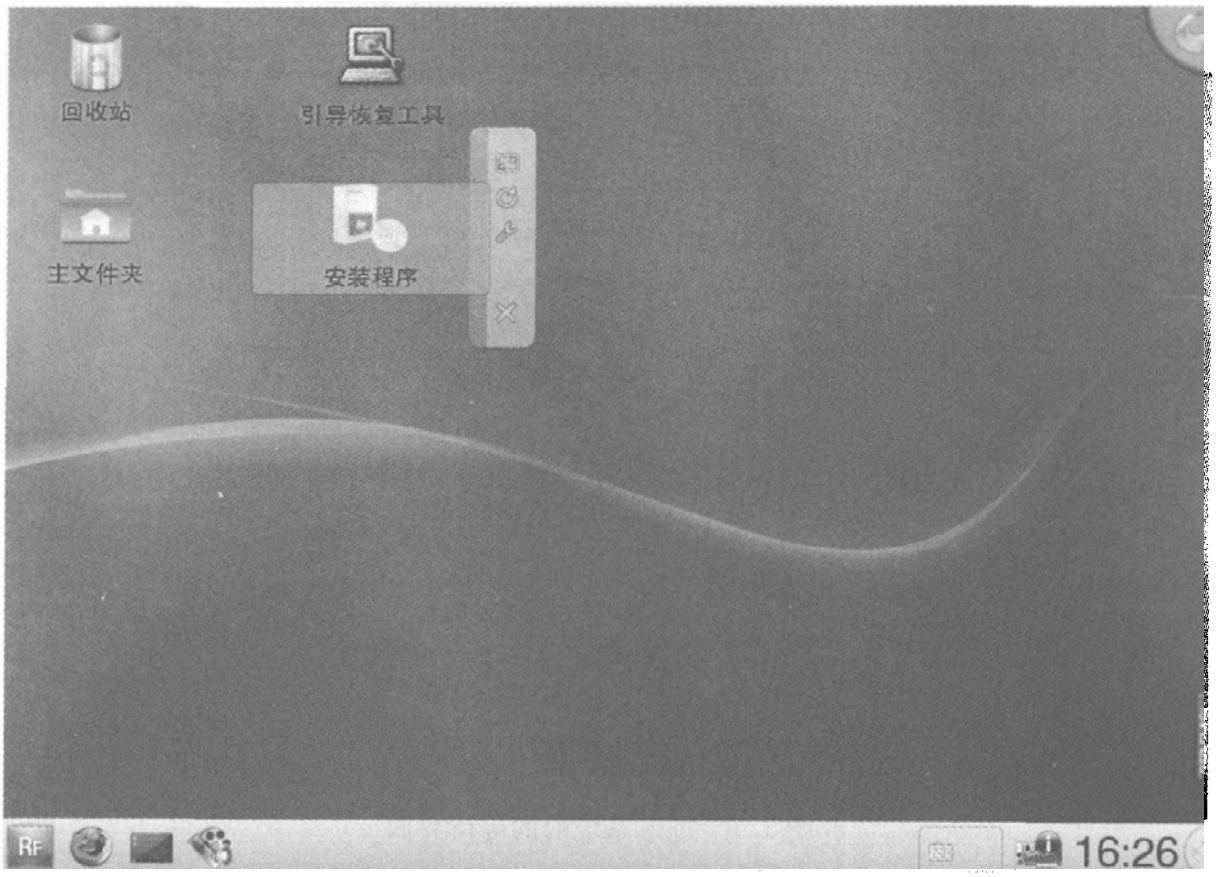


图 1.3 LiveCD 系统主界面

此时用户就可以使用系统了，不过如前所述，在此系统中用户不能保存个人文件或进行系统设置。如果需要保存，只能将文件保存到其他驱动器，如另一个 U 盘或移动硬盘，或电脑里的硬盘。

如果要将 Red Flag Linux 桌面 7.0 安装到硬盘，只需点击桌面上的“安装程序”图标即可进入系统安装程序向导(图 1.4)。

点击“下一步”，需要用户选择“接受红旗 Linux 证书”协议(图 1.5)。

点击“下一步”，进入“分区方式”界面，在选择分区方式时，初学者用户可以选择“自动方式”或“简易方式”；熟练的用户可以选择“高级方式”进行手动分区。此处我们采用“简易方式”，需要手动分区的用户可以参考相关的 Linux 教程或用户手册。

点击“下一步”，此时系统会显示当前的分区方式是“简易方式”，但是“下一步”按钮是灰色的，还无法点击(图 1.6)。

如图 1.7 所示，双击分区的状态栏“/dev/sda”，此时系统会弹出一个警告提示框，提示是否要为/dev/sda 创建分区表，如图 1.8 所示。

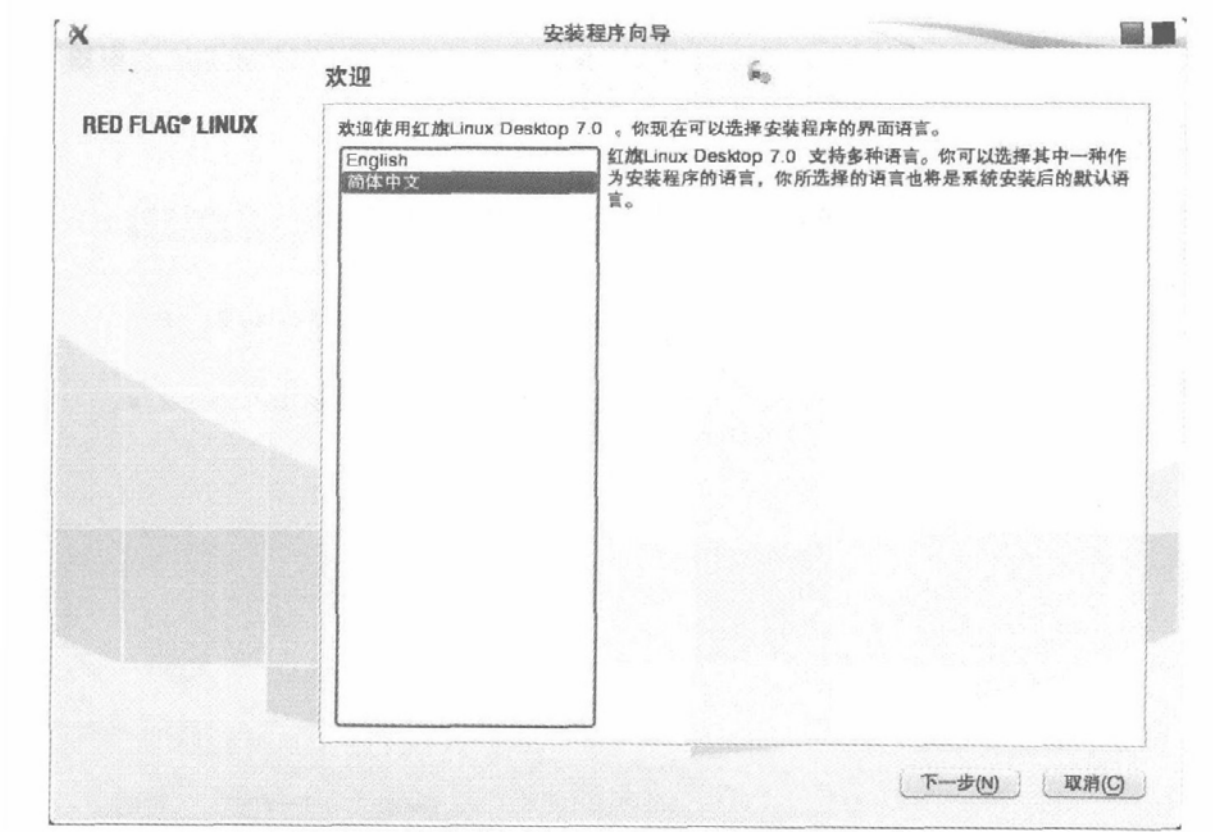


图 1.4 安装程序向导

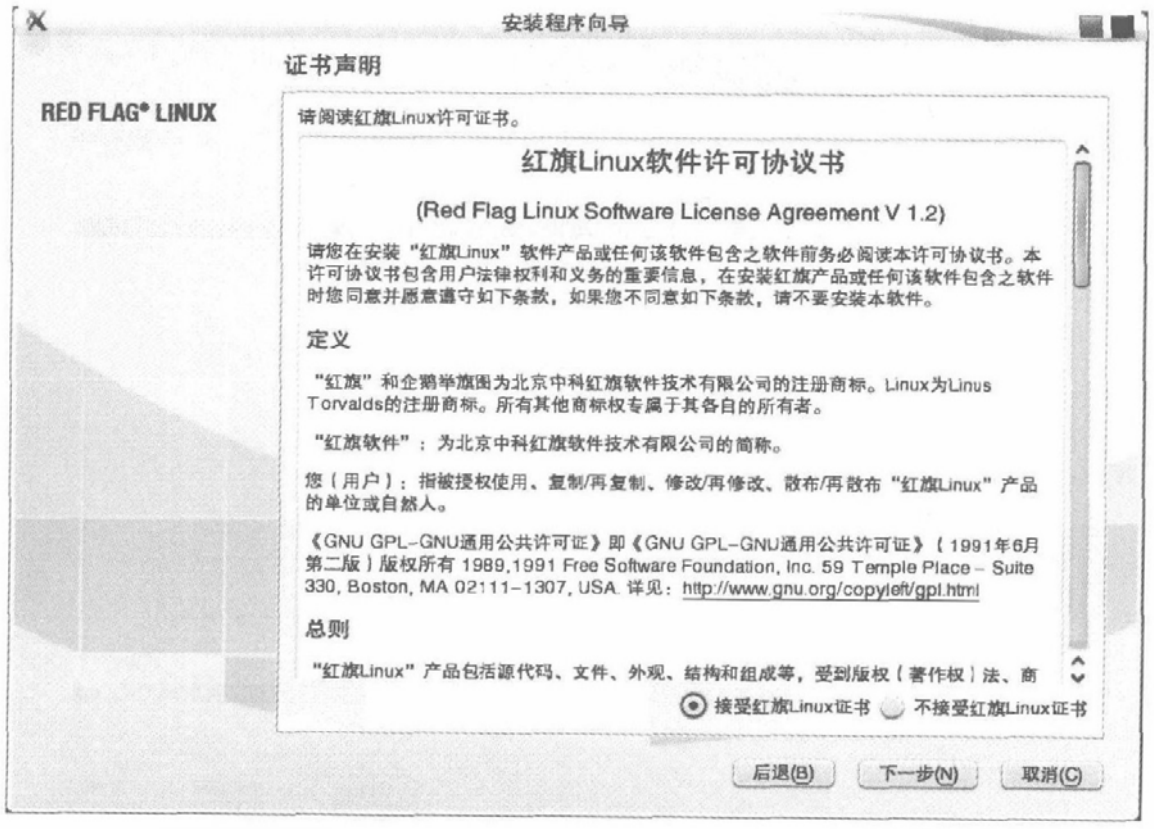


图 1.5 红旗 Linux 软件许可协议书

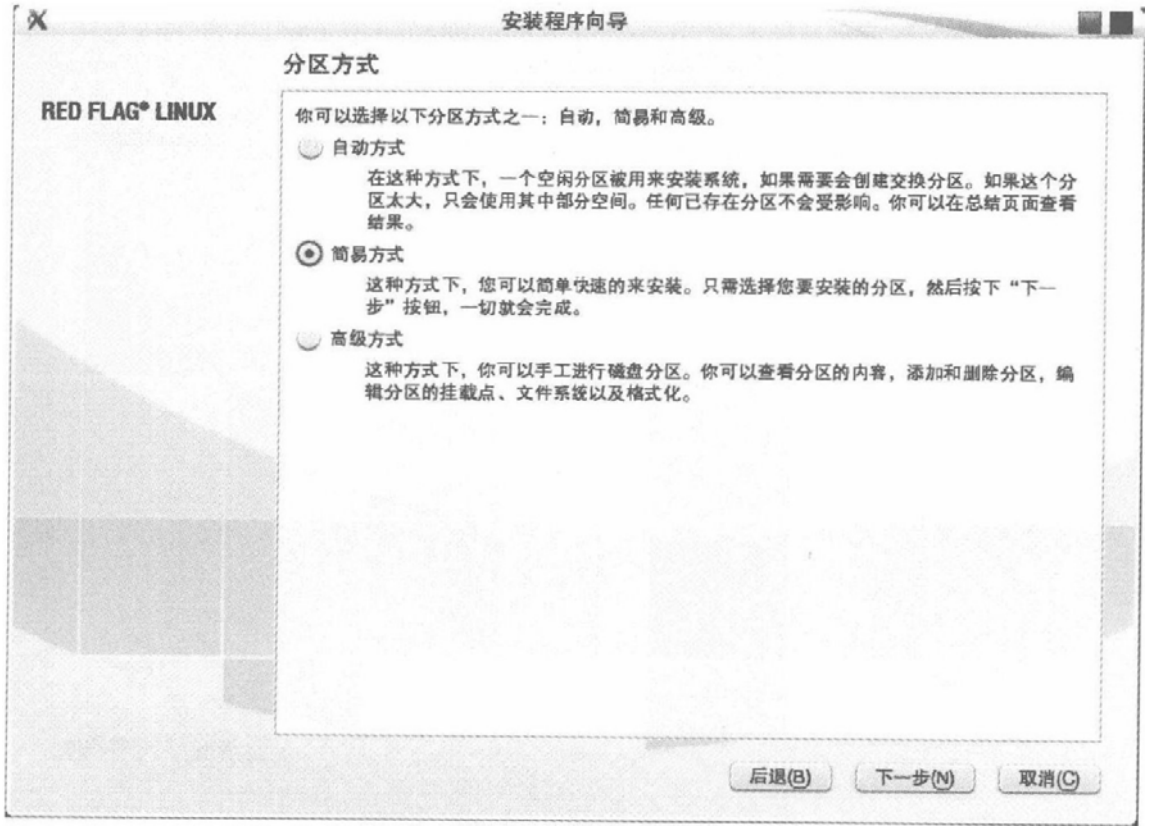


图 1.6 分区方式

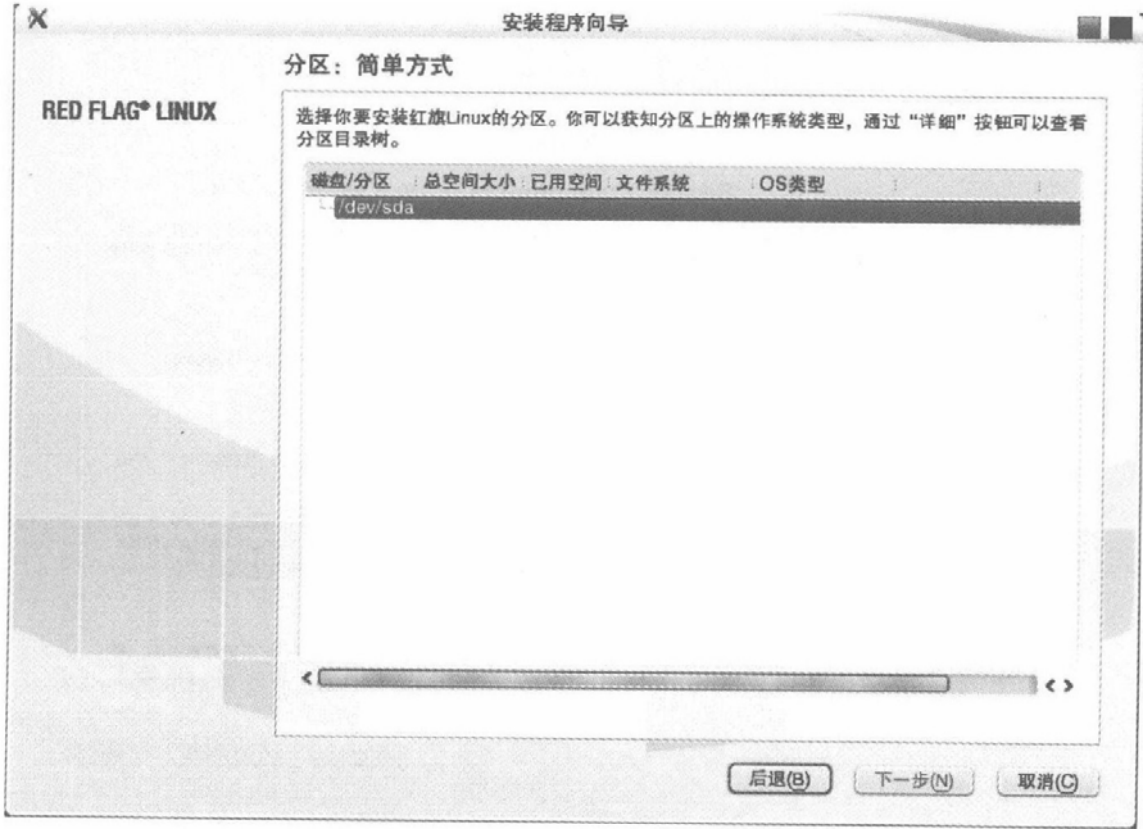


图 1.7 简易分区方式

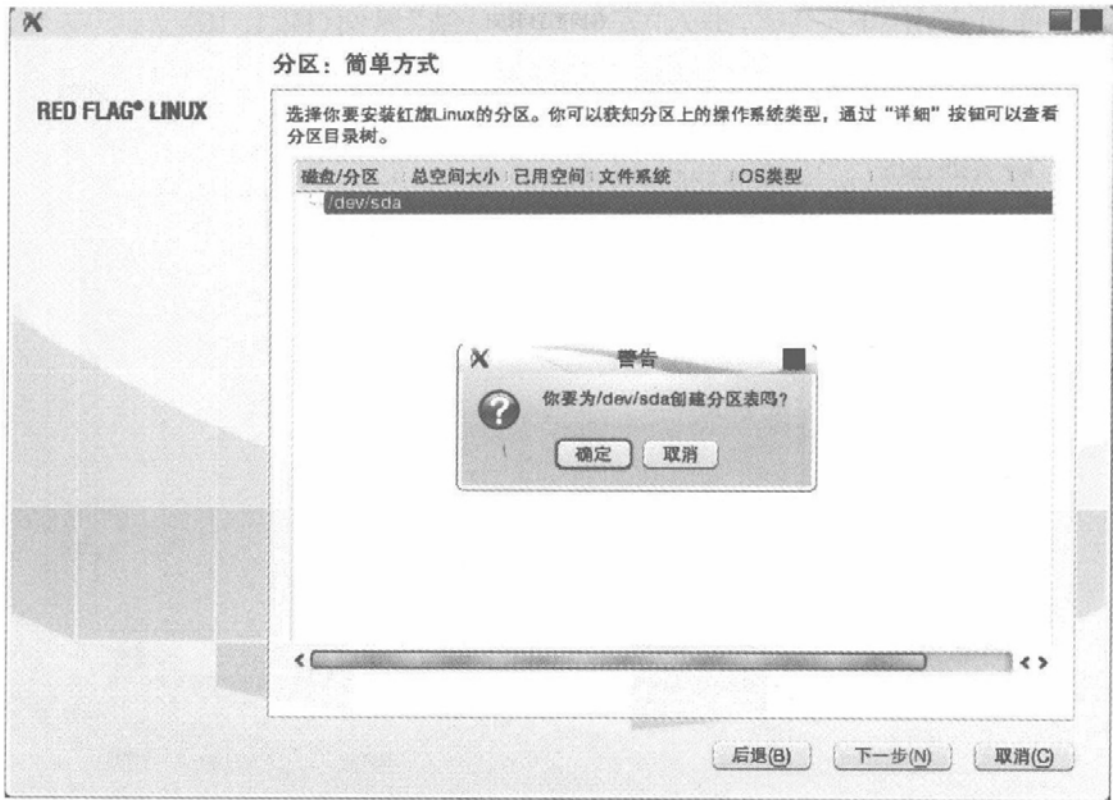


图 1.8 创建分区表提示

点击“确定”，为/dev/sda创建分区表，便可显示分区表，如图 1.9 所示。

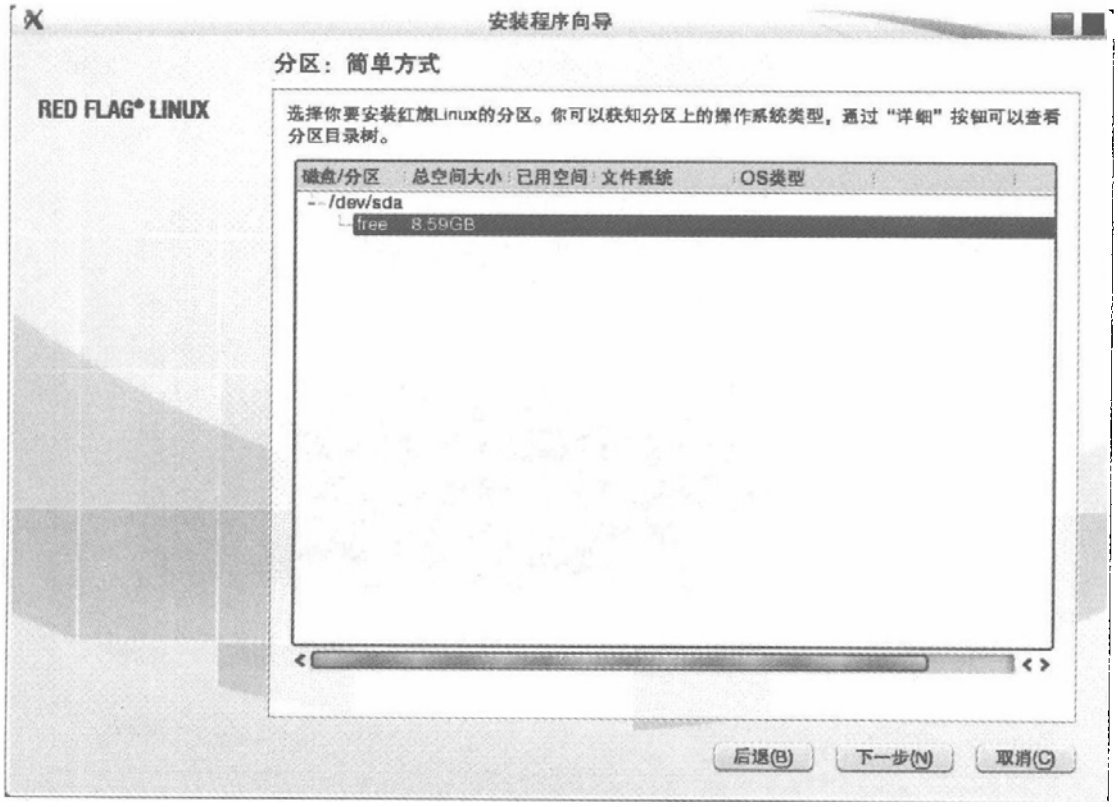


图 1.9 显示分区表

此时便可点击“下一步”,进入“安装前的总结”界面(图 1.10)。

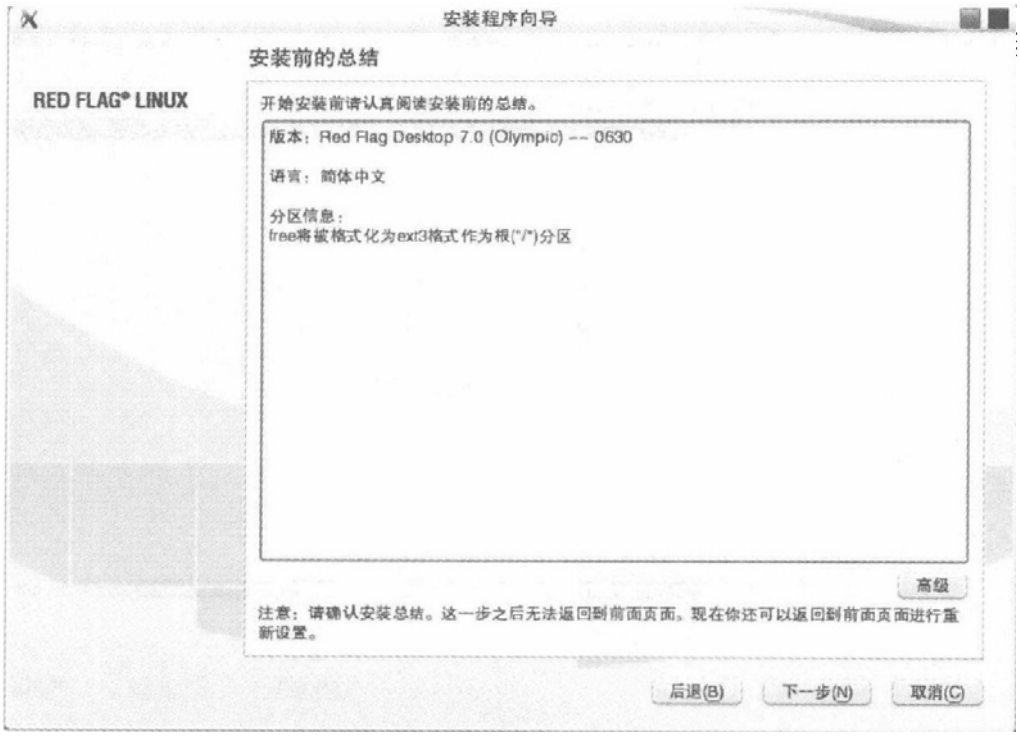


图 1.10 安装前的总结

点击“下一步”,安装 Red Flag Linux 桌面 7.0,在界面上将会显示系统的安装进度(图 1.11)。

图 1.11 安装进度

等待一段时间后，系统会提示安装过程已结束的界面(图 1.12)。

图 1.12 安装结束

点击“下一步”，进入“用户设置”界面，系统提示需要设置 root 用户密码、添加一个用户以及设置主机名，如图 1.13 所示。

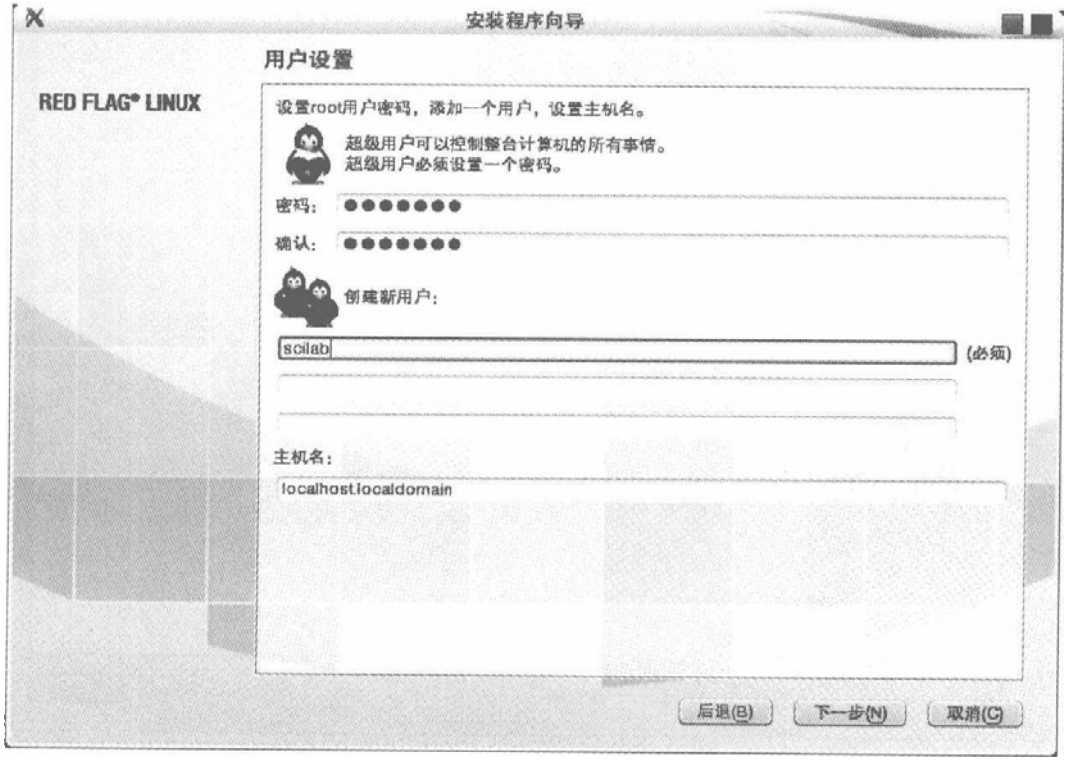


图 1.13 用户设置

点击“下一步”，系统进行安装配置，如图 1.14 所示。

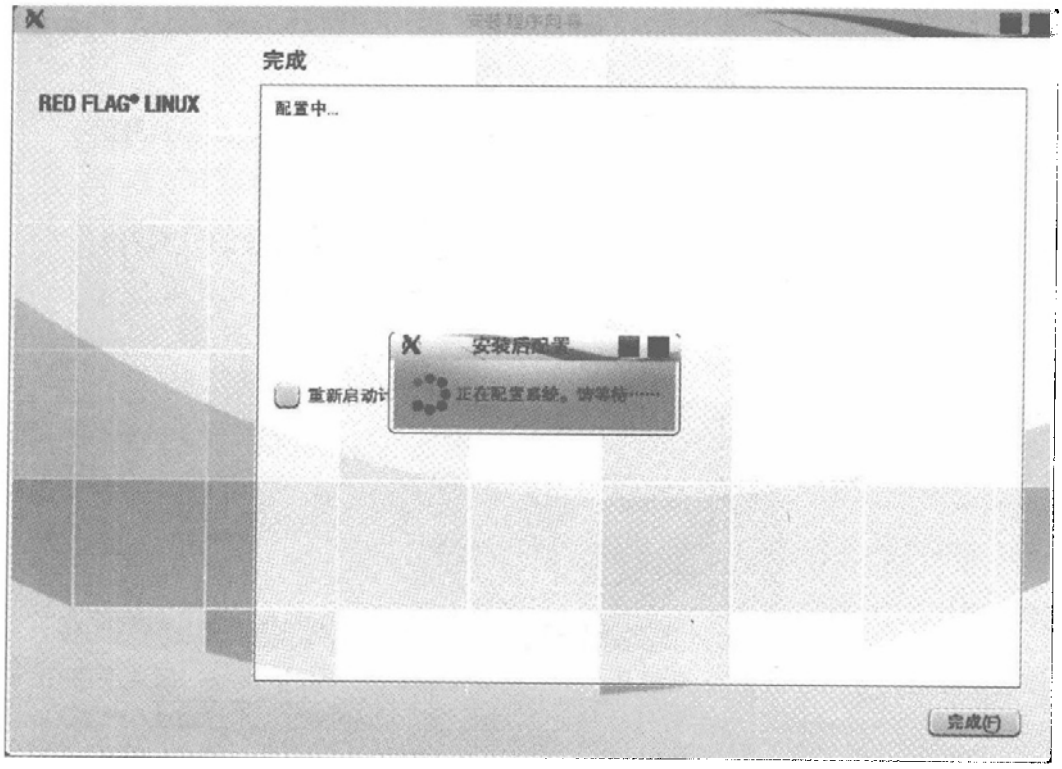


图 1.14　安装后配置

稍稍等几秒钟后，系统便会提示安装完成，如图 1.15 所示。

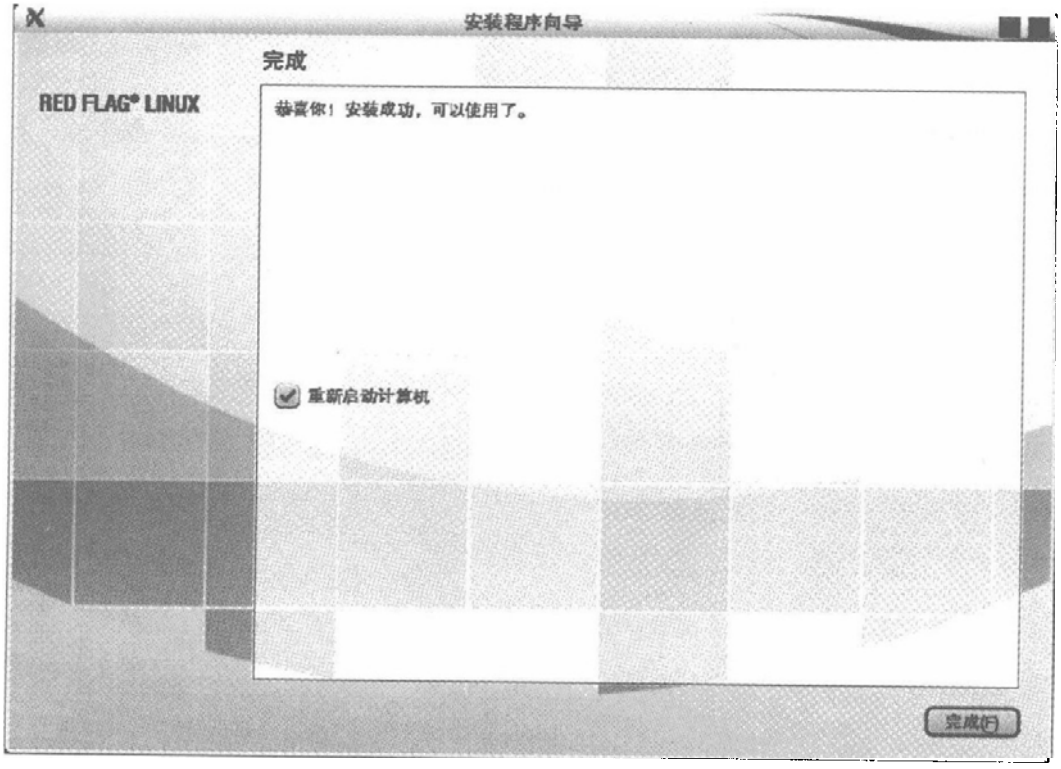


图 1.15　安装成功

选择“重新启动计算机”后，点击完成便可重新启动计算机。此时 Red Flag Linux 桌面 7.0 的安装也就大功告成了。

1.4 Linux的基本操作

1.4.1 启动和登录

如果安装 Linux 的计算机本身已安装了 Windows XP 系统，在后来安装 Linux 的过程中进行了正确的配置，按下计算机主机箱上的启动按钮进行启动时，显示器屏幕上的引导界面会显示出计算机中已安装的操作系统列表，供用户选择。用户可以通过上下光标键选择要启动的操作系统，并按下 Enter 键启动。如果不进行选择，在设定的时间之后会启动默认的操作系统。

如果计算机只安装了 Linux 操作系统，则打开计算机的电源时，则会直接启动 Linux 操作系统。

Linux 操作系统启动后，如果系统已经配置好了 X 窗口，就会打开图形化登录界面如图 1.16 所示。

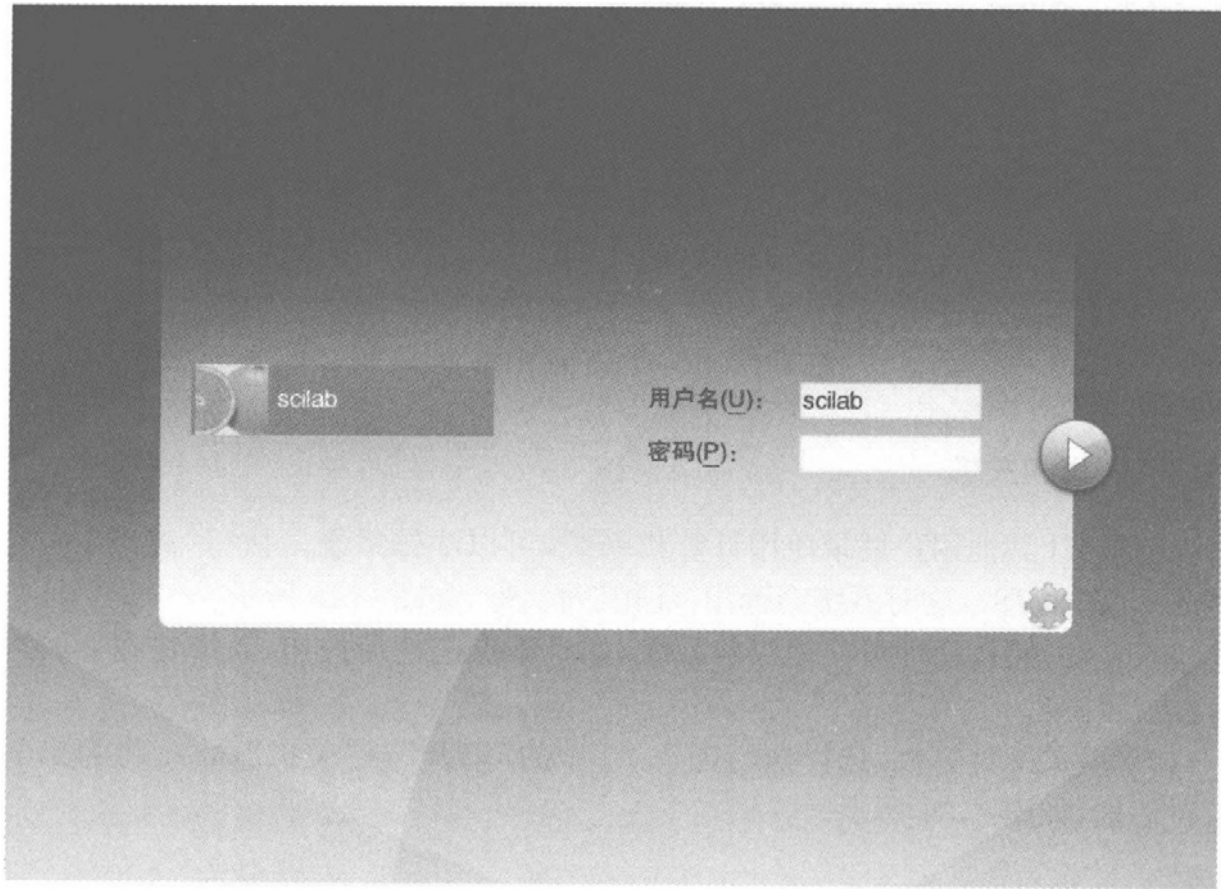


图 1.16 Red Flag 的登录界面

Linux 系统有普通用户和超级用户之分，超级用户的用户名为 root，普通用户可以在安装的时候创建也可以在登录后由系统工具创建。在登录界面中按照提示，先输入用户名，再输入密码，如果用户名和密码均正确，即可进入 Red Flag 默认的 KDE 桌面环境，如图 1.17 所示。

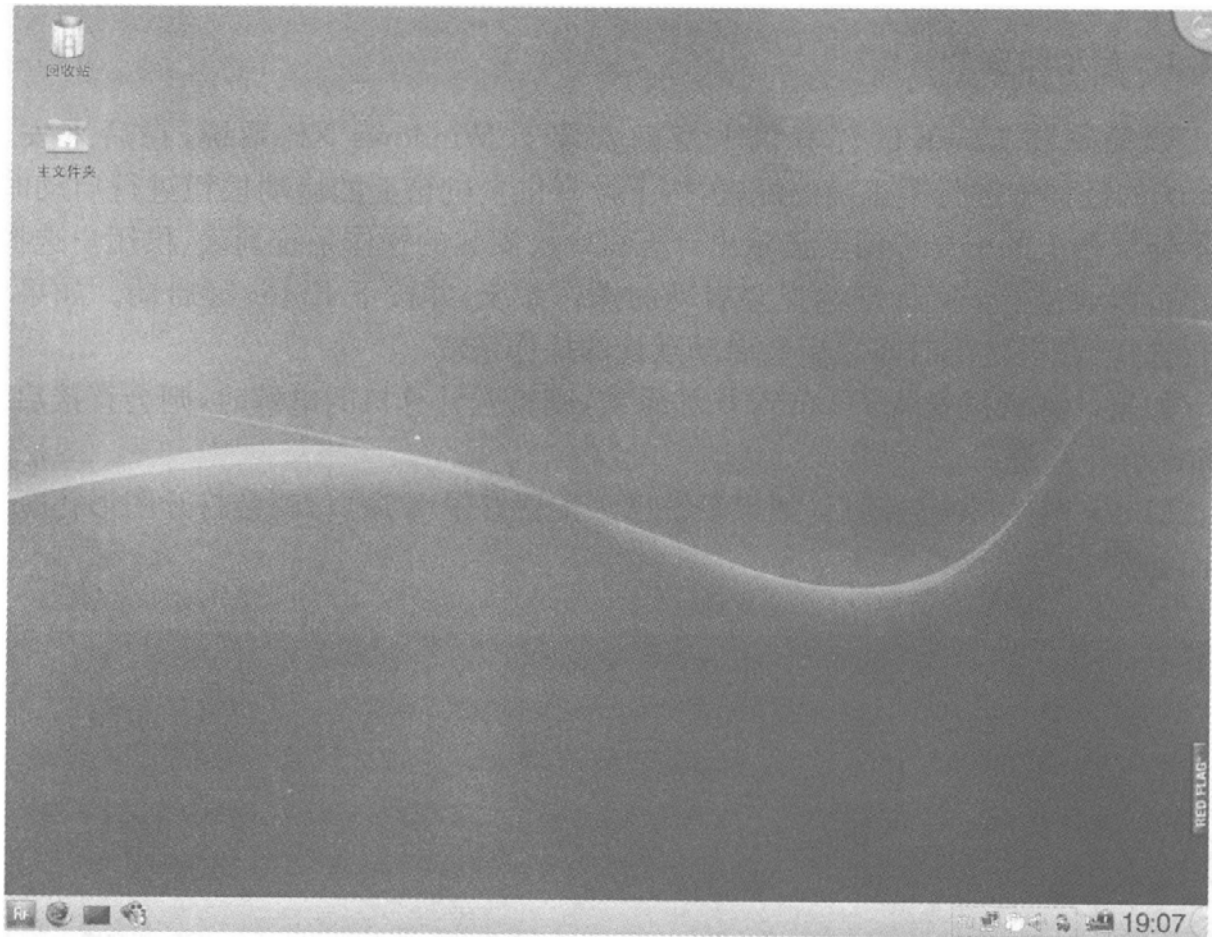


图 1.17 Red Flag 的 KDE 桌面

1.4.2 注销和关机

如果想让其他用户登录使用计算机系统，可以注销系统。执行“离开”→“注销”命令(图 1.18)，这时系统会弹出“注销”对话框，如图 1.19 所示。

单击“注销”按钮即可注销当前系统，如果不点击任何按钮，系统将在 30 秒后自动进行注销。

如果要关闭计算机，选择执行图 1.20 中的“离开”→“关机”命令，将弹出“关机”对话框，如图 1.21 所示。

图 1.18 桌面菜单中的注销命令

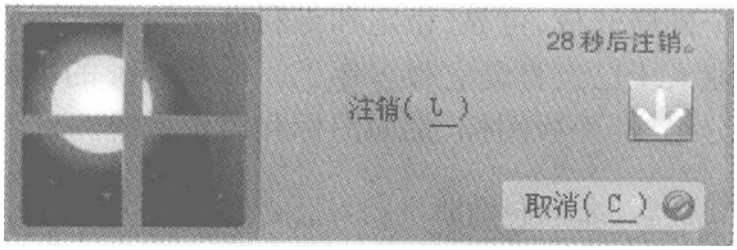


图 1.19 注销对话框

图 1.20 桌面菜单中的关机命令

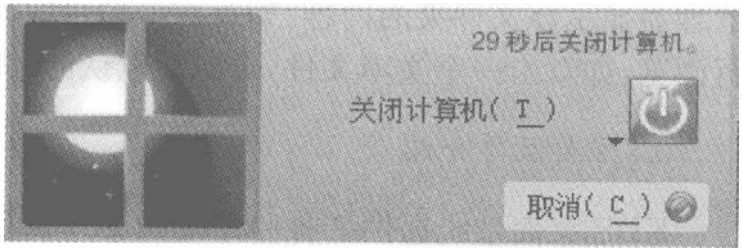


图 1.21 关机对话框

单击“关机”按钮即可关闭当前系统，如果不点击任何按钮，系统将在 30 秒后自动关闭。

在对话框中单击“重新启动”按钮将重新启动计算机。

1.4.3 KDE 桌面环境

1. KDE 简介

KDE 现在是 Unix 上可用的易于使用的现代桌面环境。和一些如 GNU/Linux 这样的自由的类 Unix 一起，Unix/KDE 组成了一个对于任何人都可用的完全自由和开放的计算平台，而且完全免费，任何人都可以修改它的源代码。当然它总是有可以改进的空间，目前已经发布了一些当今可用的能和商业操作系统/桌面组合的合适的替代品。希望 Unix/KDE 组合将会最终为普通计算机用户带来一个同样开放、可靠、稳定和专利自由的计算环境，世界范围内的科学家和计算机专业人士已经喜爱它很多年了。Red Flag 的桌面环境（如图 1.17 所示）主要由三部分组成：面板图标、桌面图标和菜单系统。

2. 桌面和面板

横贯桌面底部的长条叫做面板（top panel），它是 KDE 界面和行为的中心，Red Flag Linux 版本的面板默认在桌面的最底层。面板的位置可以移动，如有需要可以将其拖动到桌面的左方或右方。图 1.22 所示为 KDE 面板，上面包括[redacted]按钮、常用的应用程序快捷图标以及显示系统当前运行应用程序的任务条。

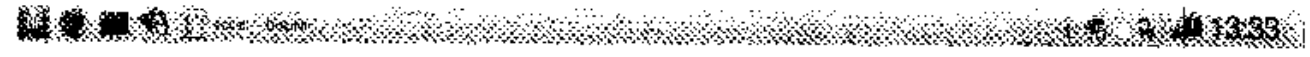


图 1.22 KDE 面板

屏幕上除了面板之外的区域就是桌面。可以把常用的程序或文件、目录（又称为文件夹）放在桌面上。默认情况下，桌面上有两个图标（如图 1.17 所示）：“回收站”图标和用户主文件夹图标。其中用户主文件夹图标相当于 Windows 中的“我的文档”图标。双击应用程序图标可运行该应用程序；双击文件夹图标可打开文件管理器显示该文件夹的内容；如果双击的是数据文件，则会打开对应的应用程序进行浏览，如双击一个文本文件，会启动默认的编辑器并打开该文本文件。

3. 主要菜单介绍

KDE 桌面系统提供了各种方便的菜单供用户使用，主要包括系统菜单、控制菜单、窗口菜单和快捷菜单等。

“系统程序”菜单位于面板的最左端，相当于 Windows 中的“开始”菜单中的程序子菜单，其中列出了系统中安装的主要应用程序，不同的是，KDE 将这些应用程序根据应用种类分成了若干大类，分别用子菜单如收藏夹、应用程序、计算机、最近

使用和离开等进行组织(如图 1.23～图 1.25 所示),方便用户浏览并启动所需的应用程序。可以单击面板上的按钮或者使用 Alt+F1 快捷键打开。

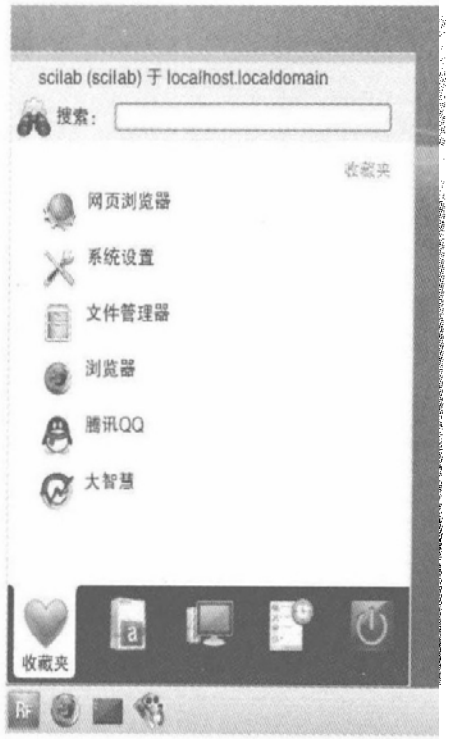


图 1.23 收藏夹菜单

图 1.24 应用程序菜单

图 1.25 计算机菜单

“控制”菜单如图 1.26 所示，几乎所有的窗口都提供控制菜单，用来执行恢复、移动、最大化、最小化窗口、配置窗口行为以及关闭窗口等操作，对应的快捷键是 Alt＋F3。

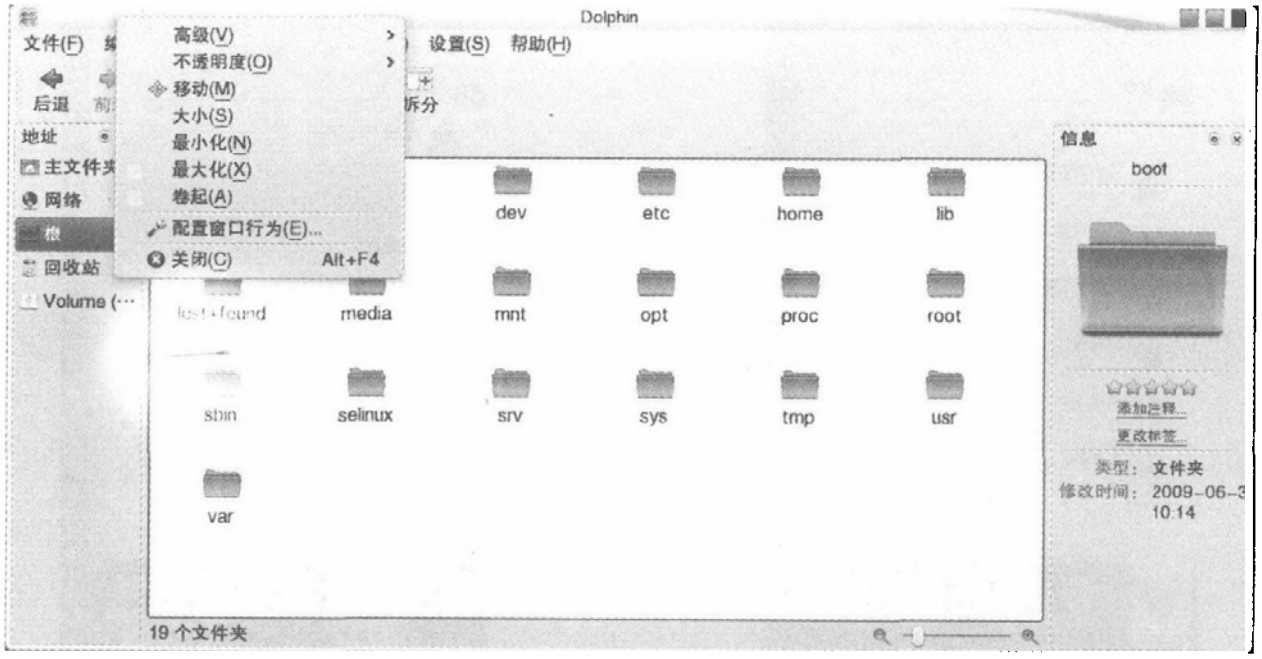


图 1.26　控制菜单

“窗口”菜单对于 Windows 用户来说是非常熟悉的，此类菜单项用来反映该应用程序的功能和可以完成的操作，常见的菜单项有“文件”、“编辑”、“查看”、“选项”、“设置”、“帮助”等，每个菜单中又包括许多子菜单项。

“快捷”菜单是另一个重要、常用的菜单。在桌面上的任何位置右击鼠标，系统就会弹出“快捷”菜单，菜单的选项会随着点击位置的窗口和界面的不同而有所差异。图 1.27 所示的是在桌面的空白位置右击弹出的“快捷”菜单，该快捷菜单可以

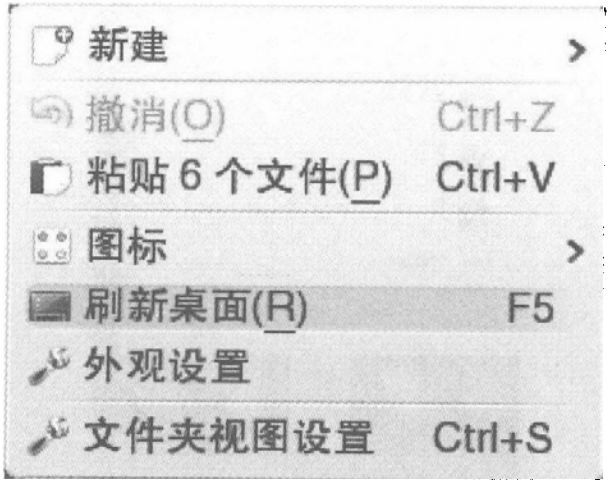


图 1.27　快捷菜单

完成创建文件夹、文件等工作，与在 Windows 中右击桌面的空白区域弹出的菜单非常类似。

4. 运行应用程序

Red Flag Linux 系列产品致力于提高 Linux 系统的灵活性和易用性，其集成了大量的应用程序，可以用来处理许多常见的任务。表 1.1 中列出了一些常用的软件。

表 1.1 Linux 中的常用软件

类 别	推荐的常用程序
文字处理器	OpenOffice. org Writer
电子表格	OpenOffice. org Calc
演示文稿	OpenOffice. org Impress
图像查看器	GThumb
文本编辑器	Gedit、vi
电子邮件客户	Evolution
万维网浏览器	Mozilla
PDF/PostScript 查看器	xpdf
声音	音频播放器(XMMS)，CD 播放器(GNOME CD)

在 KDE 中，启动应用程序有多种方式，常用方式如下：

(1) 从“应用程序”菜单中选择并单击菜单项；

(2) 在桌面上双击应用程序的快捷图标；

(3) 在 KDE 文件浏览器中双击可执行文件。

可见，在 KDE 中启动应用程序的方法与在 Windows 中非常相似。

5. 设置时间和日期

时间和日期属性工具允许用户改变系统日期和时间、配置系统使用的时区以及设置网络时间协议守护进程来与时间服务器的系统时间保持同步，但是，用户必须拥有系统根用户的密码。

执行“计算机”→“系统设置”→“日期和时间”命令，打开“日期和时间-系统设置”对话框，根据需要进行设置，单击“确定”按钮后即可生效。“日期和时间-系统设置”对话框如图 1.28 所示。

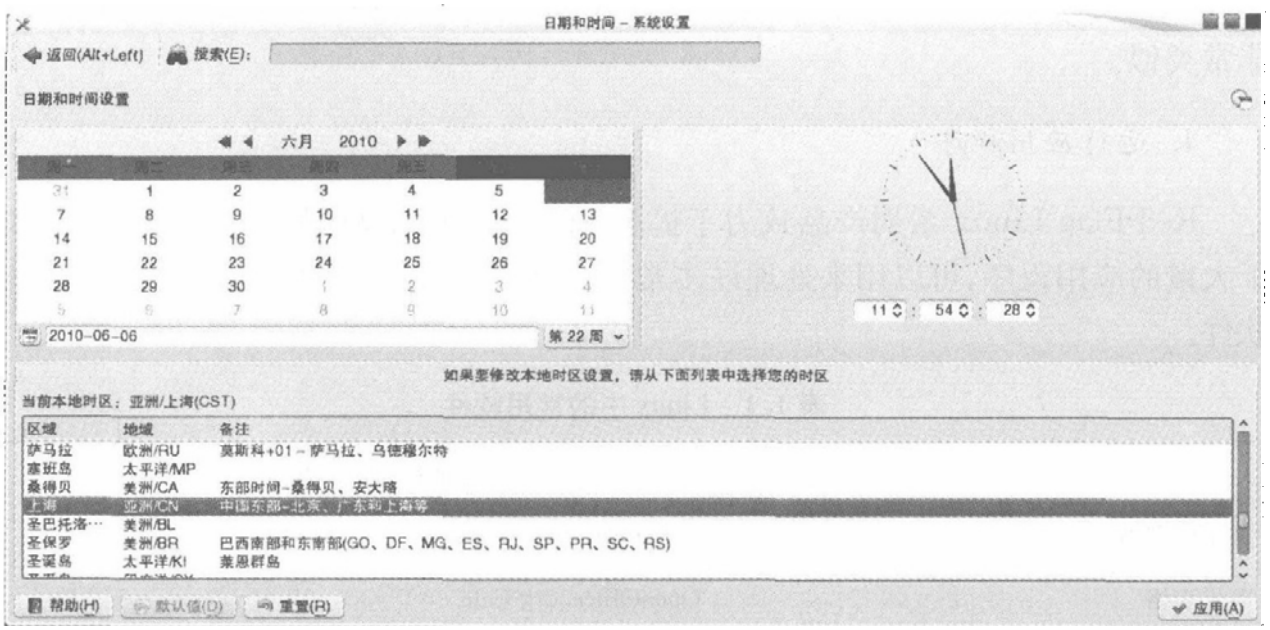


图 1.28 日期与时间设置对话框

6. 文件管理

KDE 桌面环境包括一个叫做 Dolphin 的文件管理器，它支持复制、移动和删除等基本文件操作，还有设置权限和显示项的操作。Dolphin 还提供了一些其他功能，可控制文件图标的尺寸以及定制 Dolphin 的显示外观。

启动 Dolphin 有多种方法，常用的方法如下：

(1) 在“收藏夹”菜单中选择“文件管理器”命令；

(2) 双击桌面上的用户主目录图标或“回收站”图标；

(3) 双击桌面上的其他文件夹图标。

图 1.29 显示了双击了当前用户为 scilab 桌面上的“主文件夹”之后，打开的 Dolphin 文件管理器的窗口。文件夹的名字“scilab-Dolphin”显示在标题栏的中间，窗口以图标的形式显示了 scilab 文件夹中的所有文件夹和文件，如果选中某一图标，其相关信息将显示在窗口右侧的信息栏与底部的状态栏中，如果未选中任何图标，将显示该文件夹的类型与修改时间及文件夹中的项目数目。

使用 Dolphin 进行文件的操作非常便利，文件的复制、移动、删除和重命名等操作的执行步骤和方法与在 Windows 中进行同类操作时完全一样。Red Flag 的回收站与 Windows 中完全类似，可以对删除到回收站的文件恢复到原来的文件夹。

7. 查找文件

常常需要找一些不太好找的文件，此时就需要用到系统提供的搜索功能。执

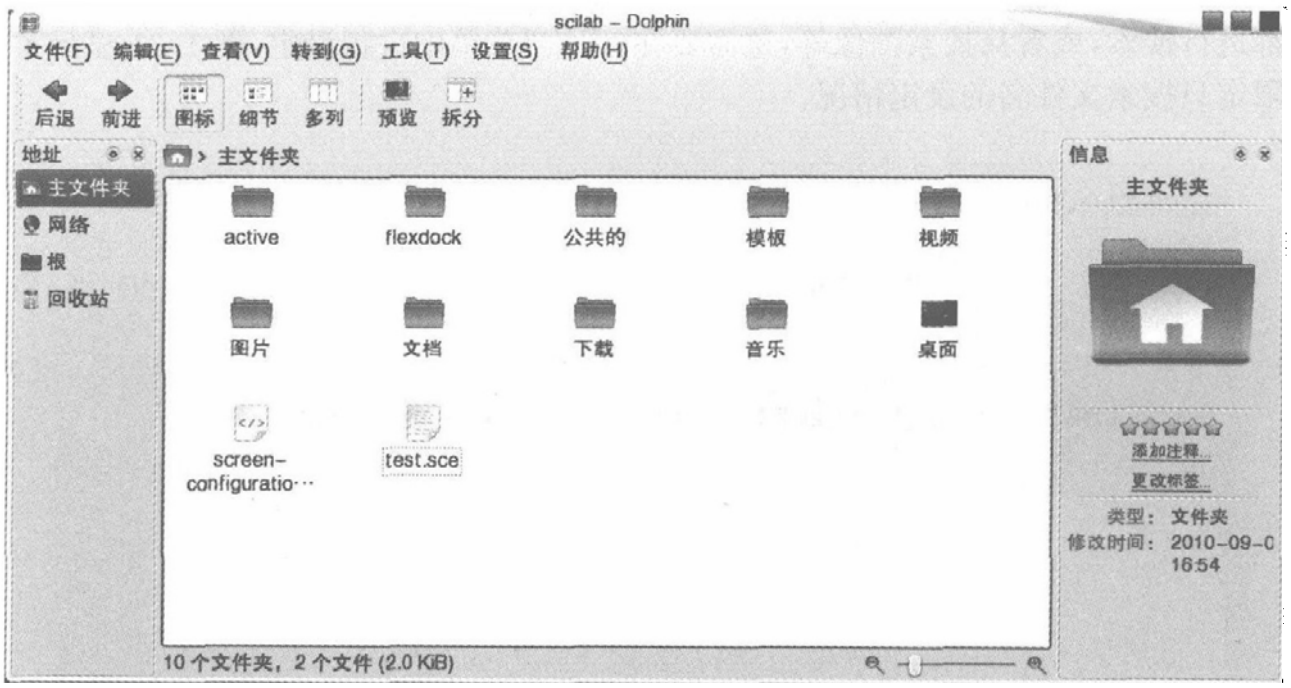


图 1.29 文件浏览器窗口

行“收藏夹”→“文件管理器”命令，系统将打开“Desktop Search”对话框，在“名为”文本框中输入要寻找的内容，点击“查找”按钮即可开始搜索。搜索的结果显示在窗口的下方，该文件的具体信息如修改时间、文件或文件夹的大小、文件所在的文件夹及权限等将全部显示在名称的右边，如图 1.30 所示。

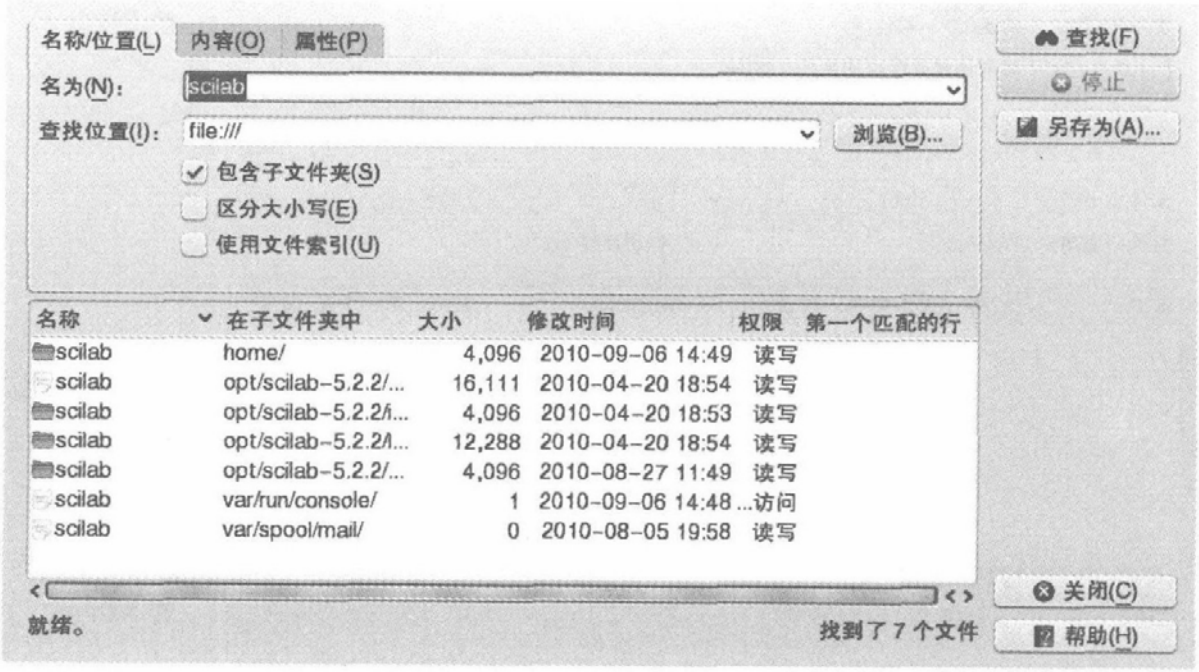


图 1.30 查找对话框

该搜索软件还可限定搜索的类型，如限定搜索文件类型是文件或文件夹内容都进行搜索，或者只搜索图像等，这需要在“内容”菜单中进行设定，图 1.31 显示了限定只搜索文件名的设定情况。

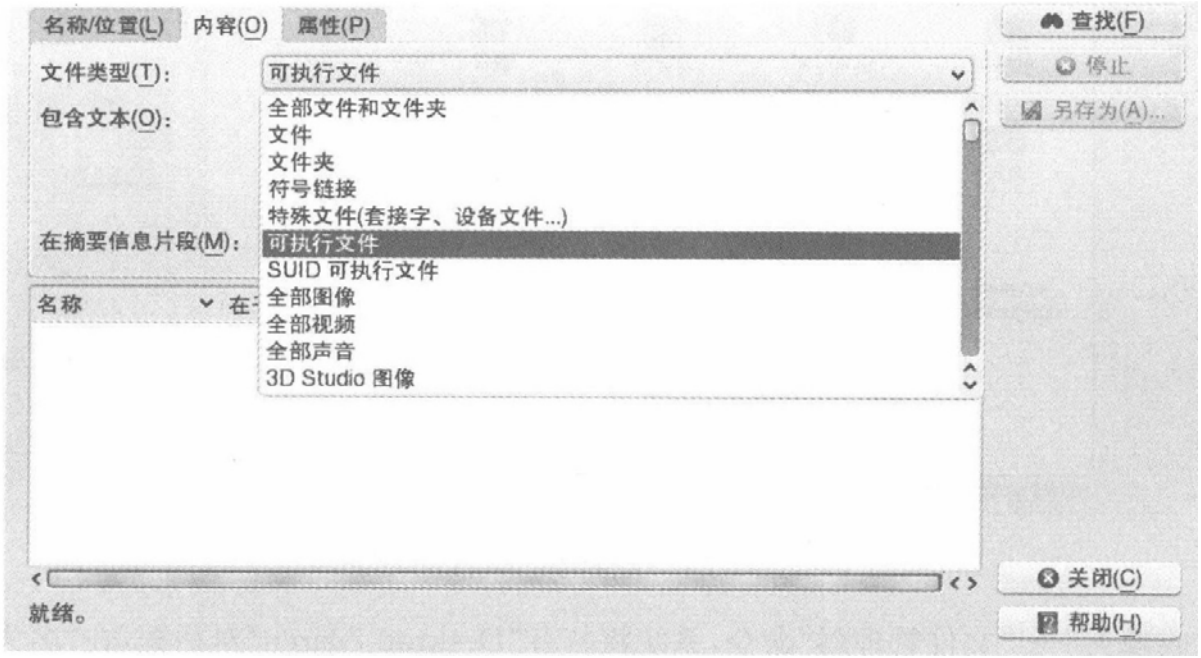


图 1.31　搜索对话框的内容设置

在文件查找对话框中还可以对所要查找的文件属性进行设置，如图 1.32 所

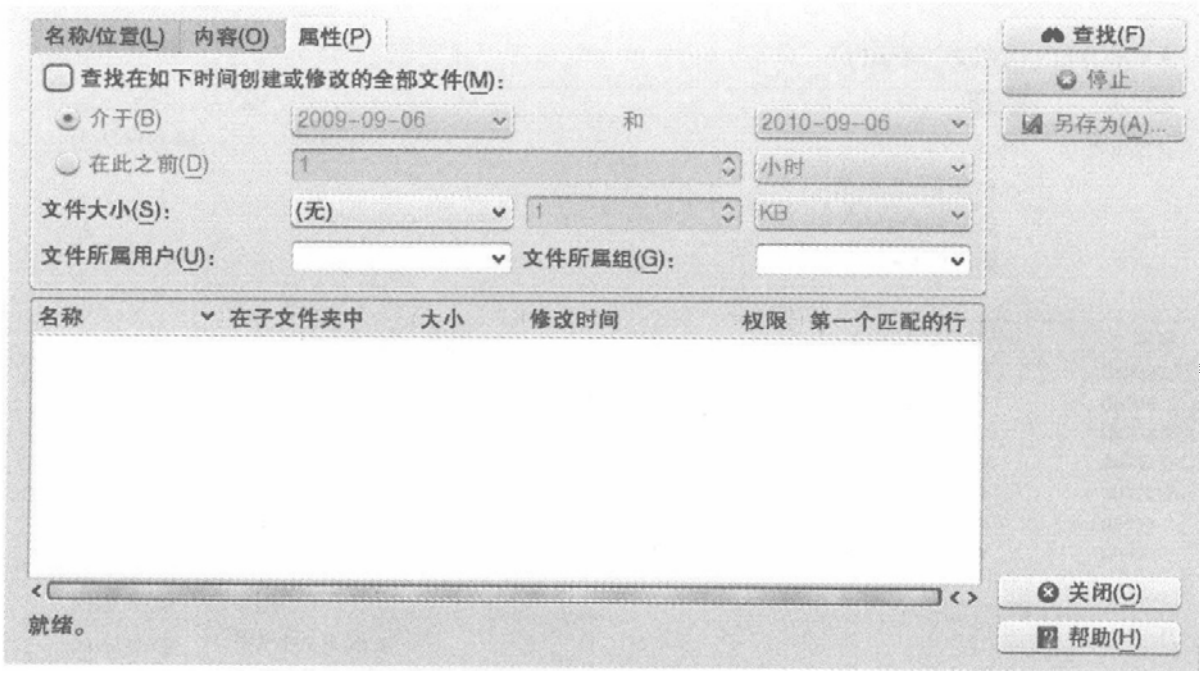


图 1.32　搜索对话框的属性设置

示，可以实现对所要查找的文件创建或修改时间、文件大小、文件所属用户以及文件所属组进行设置，实现更加精确的查找。

8. 软盘、光盘和 U 盘的使用

如果在开机前已将光盘放入光驱或者把 U 盘插入 USB 接口，则在 Linux 启动的过程中会自动挂载这些设备，访问这些设备就可以如同访问一个普通的文件夹一样，通过打开文件管理器，在 Dolphin 窗口的左侧会显示这些设备的标识，选中即可打开，如图 1.33 所示。

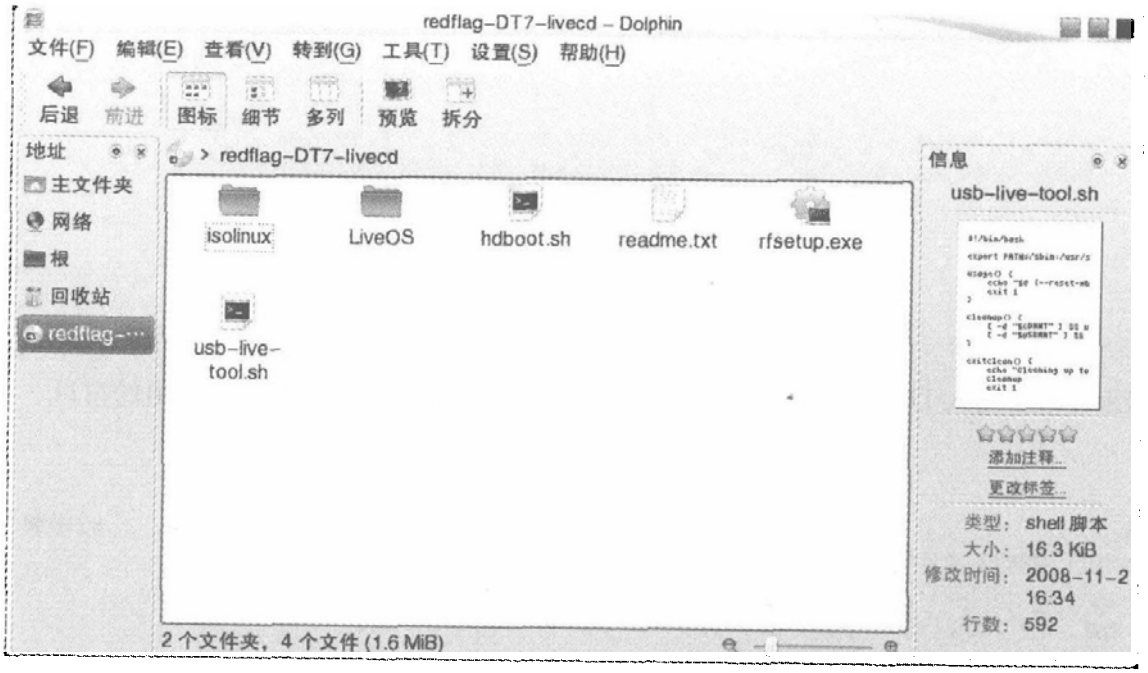


图 1.33 加载光驱后的窗口

如果在 Linux 启动之后才放入光盘或将 U 盘插入 USB 接口，这是 Linux 会自动对这些设备进行挂载，在桌面的右下方会弹出一个窗口，提示用户系统挂载了新的设备，如图 1.34、图 1.35 所示。用户只需要打开“文件管理器”，在打开的窗口的左侧选中对应的设备驱动器图标，稍等片刻即可打开。

当光盘或 U 盘使用完毕，或要换上新的设备时，需要将现有的设备卸载，此时首先要确保盘中的文件和/或文件夹均已停止使用，然后右击 Dolphin 窗口左侧对应的标识，在弹出的快捷菜单中选择弹出“redflag-DT7-livece”命令即可。图 1.36 显示了在系统中放入了光盘后，右击图标对其进行卸载时的弹出菜单。图 1.37 显示了在系统中插入了 U 盘后，右击图标对其进行卸载时的弹出菜单。用户可以看到对它们的卸载操作是完全类似的。

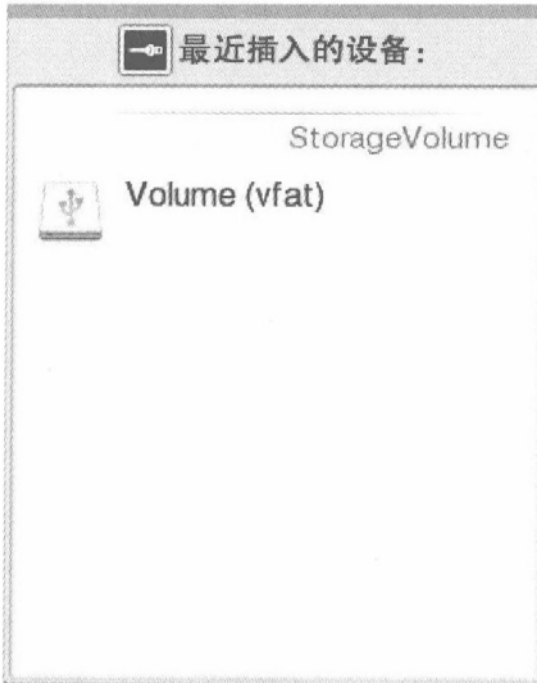


图 1.34 插入 U 盘后弹出的窗口

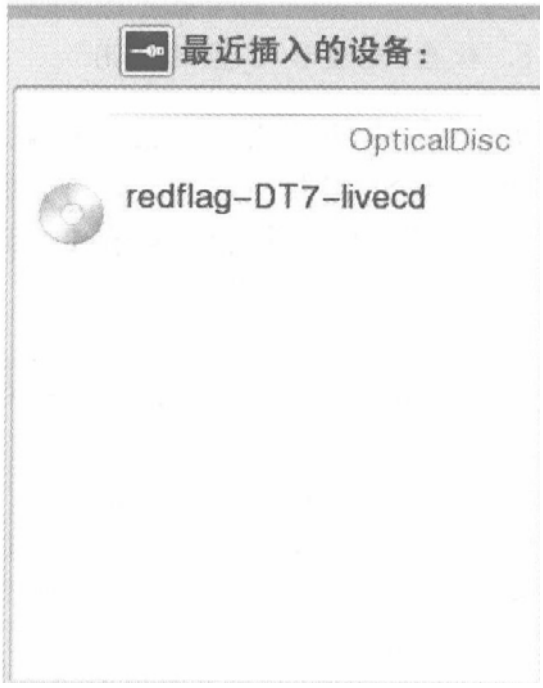


图 1.35 插入光盘后弹出的窗口

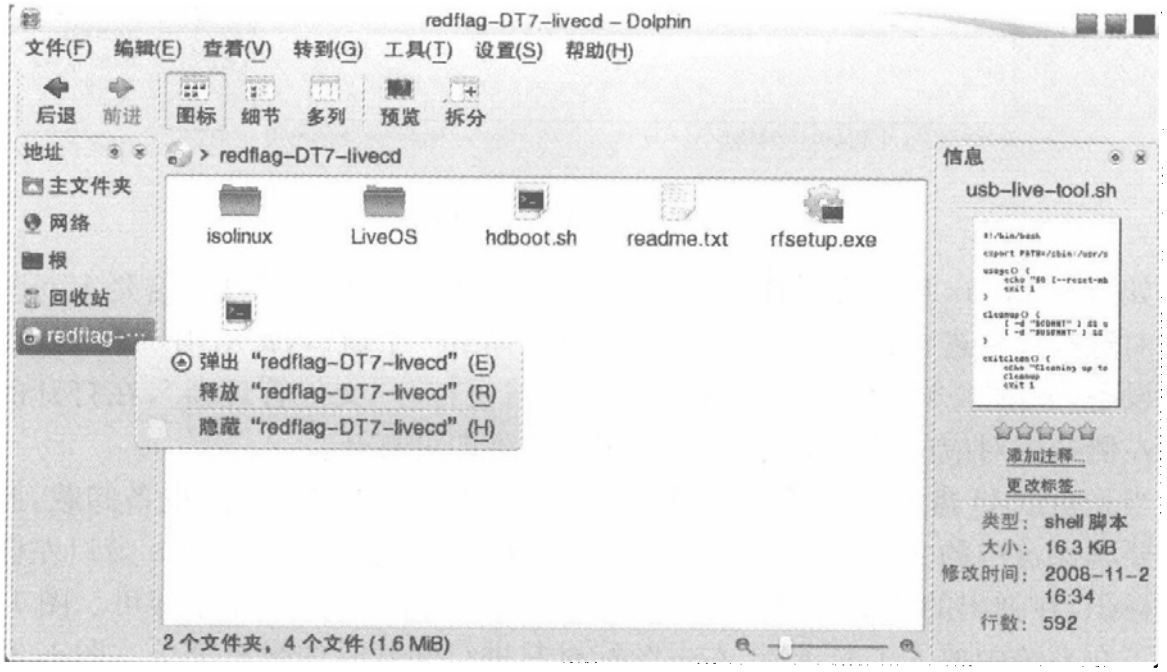


图 1.36 卸载光盘时的浏览窗口

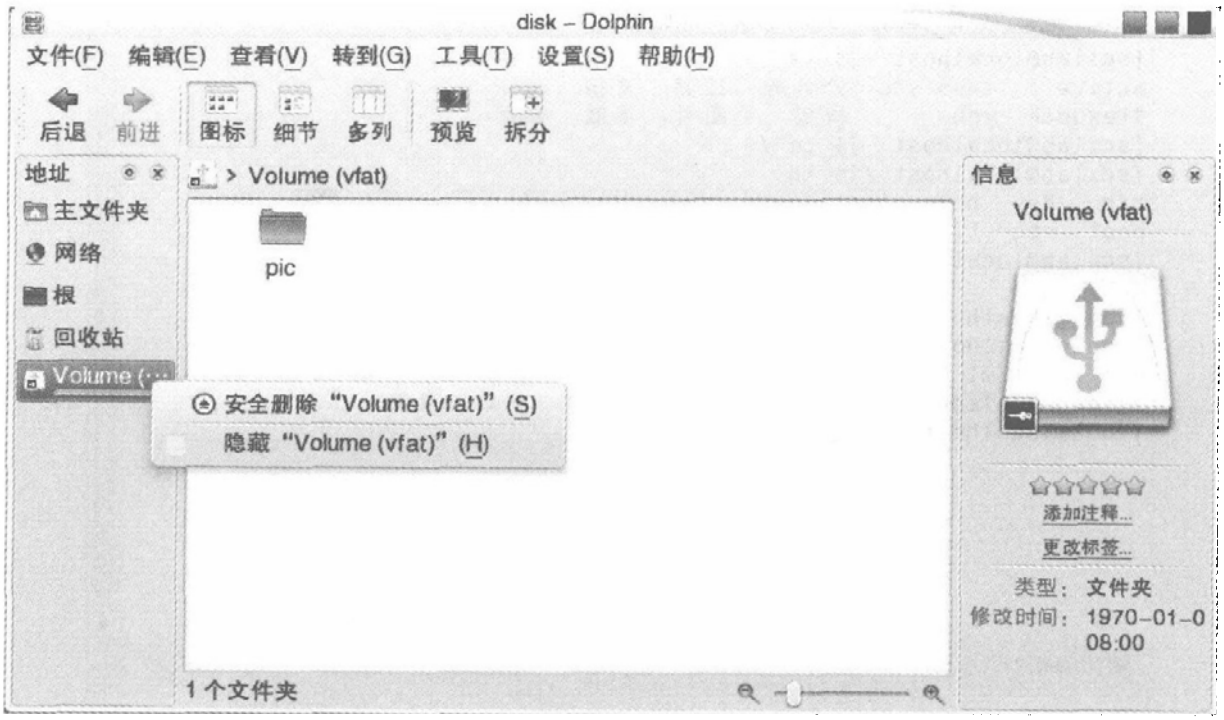


图 1.37 卸载 U 盘时的浏览窗口

1.5 终端窗口的使用

1.5.1 终端窗口

Red Flag Linux 桌面 7.0 提供了一个标准的命令行接口，这就是终端窗口。用户可以在窗口显示的提示符后输入带有选项和参数的命令并按下 Enter 键即可执行该命令。

执行“应用程序”→“系统工具”→“终端”命令，可打开终端窗口。

终端窗口被打开后，将显示一个 shell 提示符，通常为字符“$”，如果是根用户 root 登录，则提示符为“#”。在提示符后输入 Linux 命令，即可在窗口中看到命令运行的结果，后面会紧跟一个 shell 提示符，表示可以输入新的命令。图 1.38显示了输入命令“ls”之后的终端窗口。

如果想关闭终端窗口，可以在 shell 提示符后输入“exit”命令(不含引号)，也可以直接单击窗口的关闭按钮。

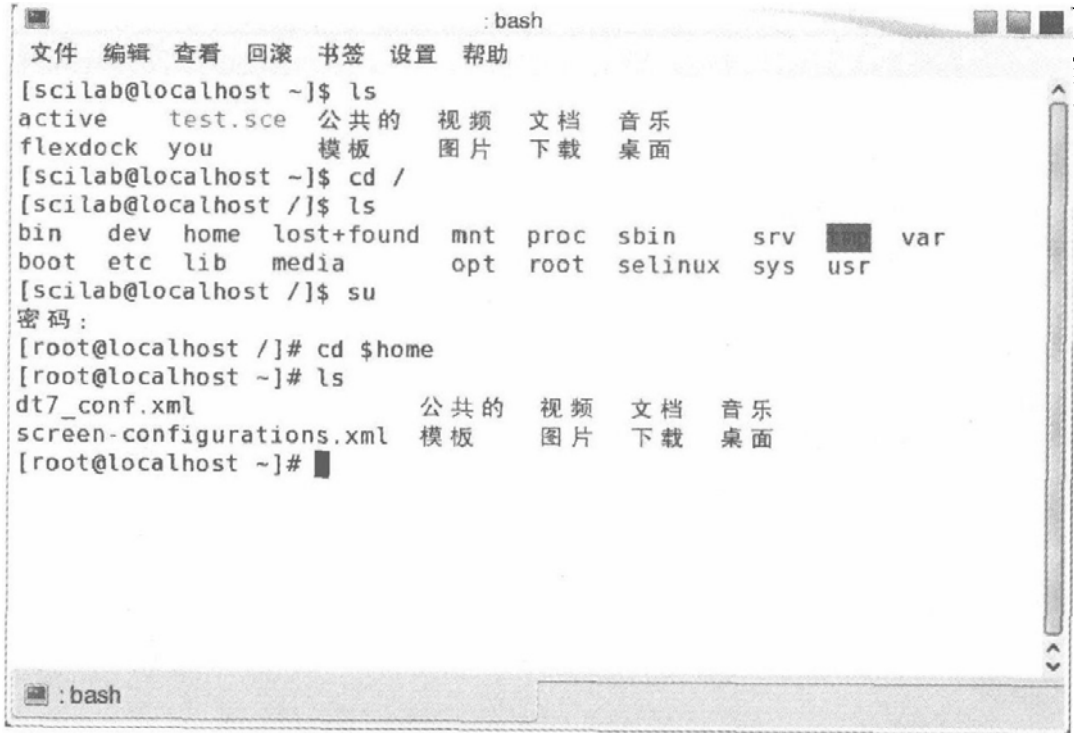


图 1.38 “ls”命令的显示结果

1.5.2 常用终端命令

表 1.2 显示了常用的终端命令。

表 1.2 常用的终端命令

命 令	功 能
clear	清除终端屏幕
cp	将文件或文件夹复制到其他文件夹
date	显示系统当前的日期和时间
diff	找出两个文件之间的不同之处
exit	关闭终端窗口
fdformat	格式化软盘
free	查看当前系统内存的使用情况
gedit	打开一个全屏幕文本编辑器
less	一屏一屏地查看超过一个屏幕的文件的内容
ls	显示当前文件夹中的文件和子文件夹列表
man	显示一个命令的详细信息

续表

命 令	功 能
mkdir	建立子文件夹
mv	将文件及文件夹移动到其他位置或更改文件和文件夹的名称
pwd	显示当前所处的文件夹的完整路径
rm	删除文件夹中的文件或文件夹
vi	打开一行编辑器编辑文本文件

1.6 OpenOffice.org 办公软件

在 Red Flag Linux 桌面 7.0 中，集成了一个功能强大的 OpenOffice.org 2.0 的日常办公应用软件系列，这些软件和微软的 Office 系列类似，但在功能上绝对不逊色。只要用过 Office 系列软件，就会很容易掌握该系列软件的使用。表 1.3 显示了 OpenOffice.org 的各个常用组件与微软 Office 产品的对照关系。

表 1.3 OpenOffice.org 组件与微软 Office 产品的对照关系

OpenOffice.org 组件	微软公司产品	功能
OpenOffice.org Writer	Office Word	文字处理
OpenOffice.org Impress	Office PowerPoint	幻灯片制作
OpenOffice.org Calc	Office Excel	电子表格处理
OpenOffice.org Draw	Windows 画图	简单的绘图工具
OpenOffice.org Math	Office 公式编辑器	公式录入和编辑
OpenOffice.org Base	Office Access	数据库和报表处理

以下分别介绍三个常用组件：OpenOffice.org Writer、OpenOffice.org Calc 和 OpenOffice.org Writer Impress。

1.6.1 OpenOffice.org Writer

OpenOffice.org Writer 是一个功能强大的文字处理器，界面直观、操作方便，可用于设计和制作含有图形、表格或图表的文本文档，且可以多种格式进行储存，如.doc、.htm、.html 以及.pdf 等。

执行“应用程序”→“办公”→“字处理器”命令，或双击顶面板上的 OpenOffice.org Writer 快速启动按钮即可启动 OpenOffice.org Writer。启动后的窗口如

图 1.39 所示。

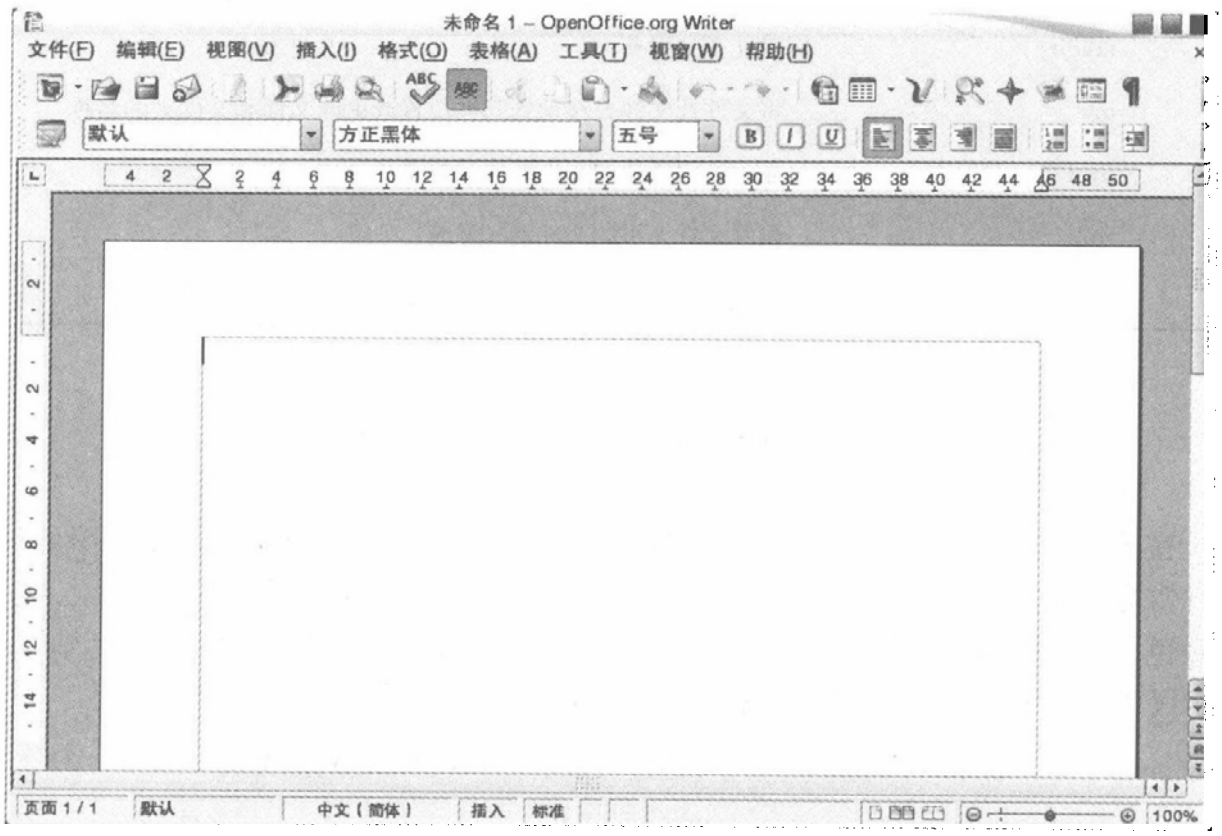


图 1.39 OpenOffice.org Writer 的界面

OpenOffice.org Writer 的界面跟微软的 Word 非常相似，用户可以很方便地找到需要的菜单命令："文件"菜单可进行文件的新建、打开和保存等文件工作；"编辑"菜单可以完成复制、剪切、插入、查找以及替换等文字编辑工作；"插入"菜单则可以在文件中插入表格、图片、影片以及声音等。当然一些常用的命令也以按钮的形式出现在菜单栏下的工具栏中，这些按钮的外观跟在微软的 Word 中对应按钮的外观可能不完全一样，但非常容易辨认。

在 Linux 中进行输入法切换的方法同在 Windows 中一样，同时按下 Ctrl、Alt 和空格键即可完成英文输入法和中文输入法的切换，也可以通过顶面板右边的输入法切换按钮（图 1.22）进行切换，可以发现 Linux 在安装的过程中也同时安装了大量的输入法。

OpenOffice.org Writer 编辑的文件的默认格式是 .odt，如果要保存为其他文件格式，选择"文件"→"另存为"命令，在如图 1.40 所示的"另存为"对话框中的"Filter"，在出现的文件类型列表中选择需要的文件类型即可。

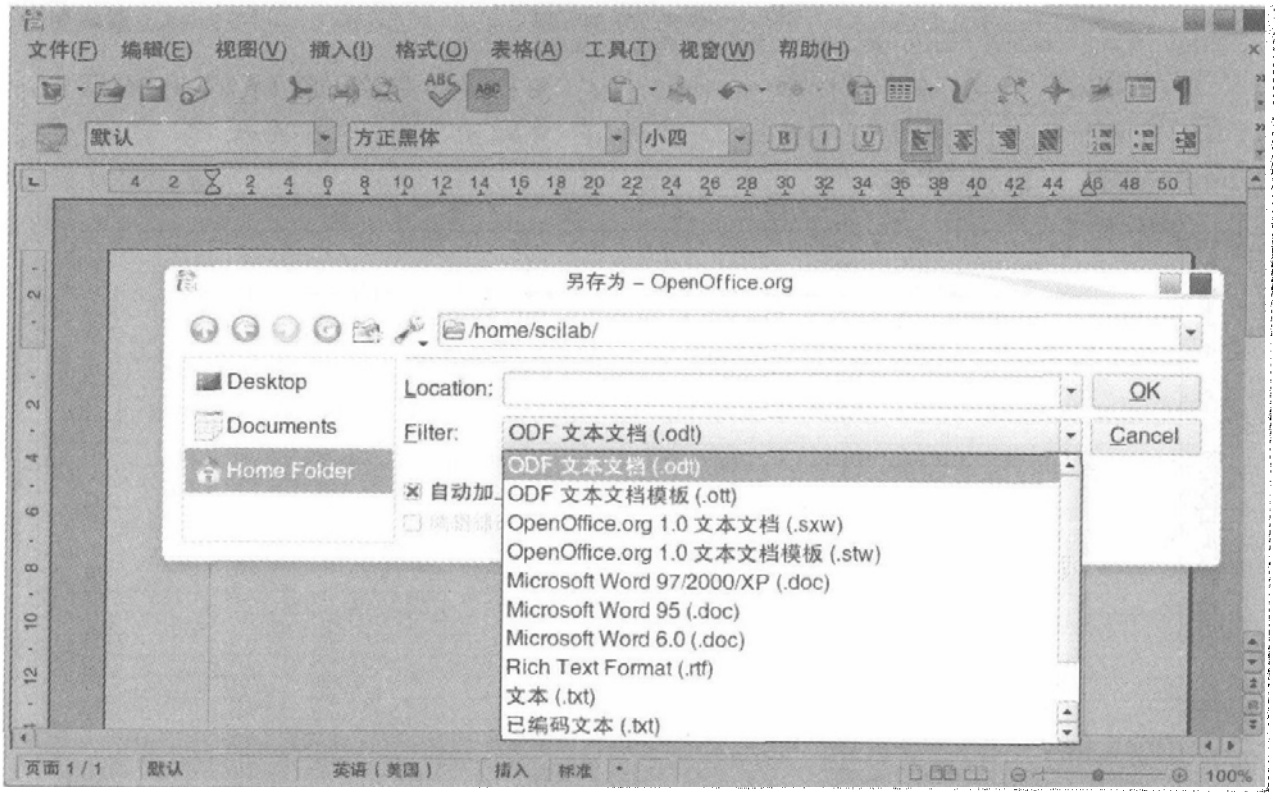


图 1.40 另存为对话框中的文件类型列表

1.6.2 OpenOffice. org Calc

OpenOffice. org Calc 是一个优秀的电子表格处理器，不仅可以进行表格处理，计算、统计以及图表制作等功能也非常出色。

执行“应用程序”→“办公”→“Spreadsheet”命令，或双击顶面板上的 OpenOffice. org Calc 快速启动按钮即可启动 OpenOffice. org Calc。启动后的窗口如图 1.41所示。

同 OpenOffice. org Writer 一样，OpenOffice. org Calc 的界面跟微软的 Excel 非常相似，常用的操作方法也比较雷同：输入数据可通过双击单元格直接输入，也可以在输入栏进行输入；输入公式时以一个“=”开头；进行快速填充，只需将鼠标放在填充起始的单元格的右下角，变为十字形状后，按下鼠标拖动到终止位置，在弹出的对话框中进行具体的设置即可（与 Excel 进行快速填充略为不同）；通过“数据”菜单可进行排序、分类汇总以及筛选等数据操作功能。

OpenOffice. org Calc 编辑的文档的默认格式是 . ods，它还可以将文档保存为其他格式如：. xlt、. xls 和 . sxc 等，此时需要用到“文件”→“另存为”命令。

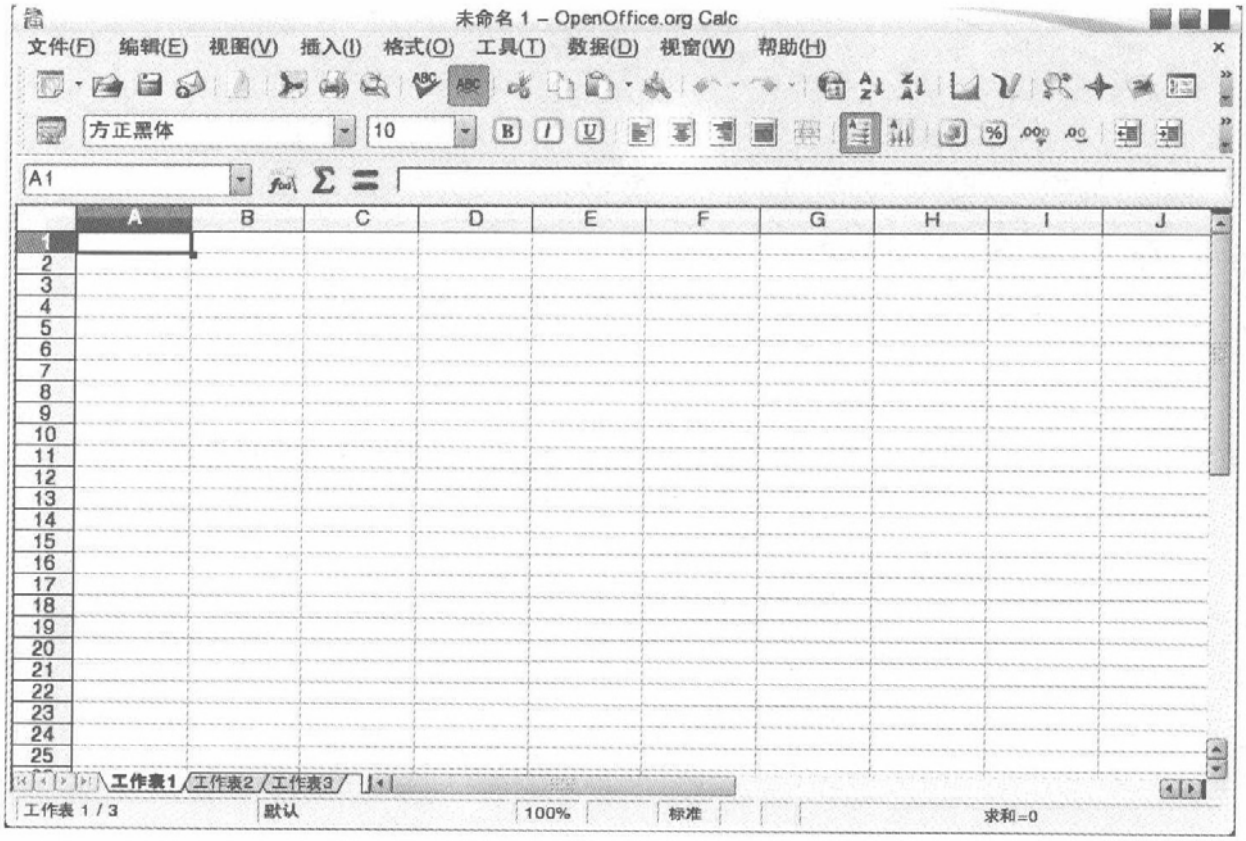


图 1.41 OpenOffice. org Calc 的界面

1.6.3 OpenOffice. org Impress

OpenOffice. org Impress 是 OpenOffice. org 系列办公软件中可以用来制作演示文稿的成员。执行“应用程序”→“办公”→“Presentation”命令,或双击顶面板上的 OpenOffice. org Impress 快速启动按钮即可启动 OpenOffice. org Impress。启动后的窗口如图 1.42 所示。

除了标题栏、菜单栏、工具栏和状态栏之外,OpenOffice. org Impress 界面的主要工作区域从左至右分为三个区域:幻灯片浏览区、幻灯片视图切换区以及任务设置区。

幻灯片浏览区显示出整个演示文稿的所有幻灯片,可以在其中方便地选择要进行编辑的幻灯片,及进行幻灯片的插入和删除等操作。

幻灯片视图切换区可进行普通视图、大纲视图、备注页视图、讲义视图以及幻灯片浏览视图之间的切换。可在普通视图中进行幻灯片的编辑工作,在大纲视图中浏览幻灯片的标题,在备注页视图中编写备注,并在讲义视图中安排讲义的打印,最后在幻灯片浏览视图中查看整个演示文稿的状况。

用户可以通过选择“演示文稿”→“放映演示文稿”来随时预览演示文稿。文稿演示采用全屏模式,可以等待演示完毕后自动退出,或者按下 ESC 键来强制终止

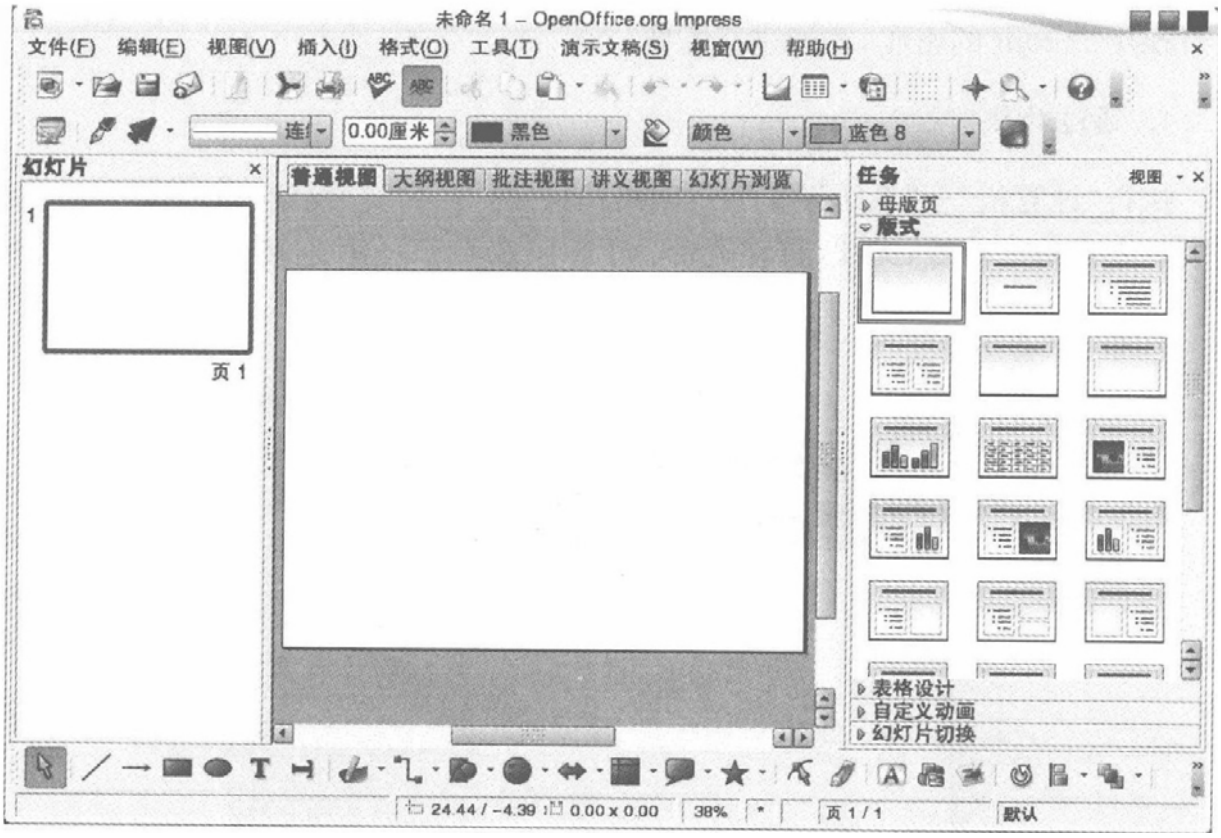


图 1.42 OpenOffice. org Impress 的界面

播放。

任务设置区可根据当前的视图模式完成幻灯片版式的设置、讲义的布局、幻灯片切换方式以及进行动画设置等工作。

选择“插入”→“幻灯片”命令可以为正在编辑的演示文稿添加新的幻灯片。

使用“文件”→“另存为”命令可以将 OpenOffice. org Impress 编辑的文档保存为其他格式如:. ppt、. pot 和 . sda 等。OpenOffice. org Impress 编辑的演示文稿的默认格式为 . odp。

1.7 Internet 的配置和 Web 浏览

1.7.1 Internet 的配置

必须进行 Internet 配置与 Internet 建立连接才能够访问网络,Red Flag Linux 桌面 7.0 提供了多种方式接入 Internet,包括专线连接、局域网连接、无线连接以及电话拨号连接等几种。不同的连接方式,所要求的软件配置与硬件配置均不相同。

下面分别介绍通过局域网方式和 DSL 接入网络的 Internet 配置。用户需要拥有根用户 root 的密码，才能进行配置。

1. 通过局域网接入 Internet

执行“计算机”→“系统设置”→“网络连接”命令，打开“网络连接”窗口，会看到在窗口中显示了有线、无线、移动宽带、VPN 以及 DSL 五种连接方式，如图 1.43所示。

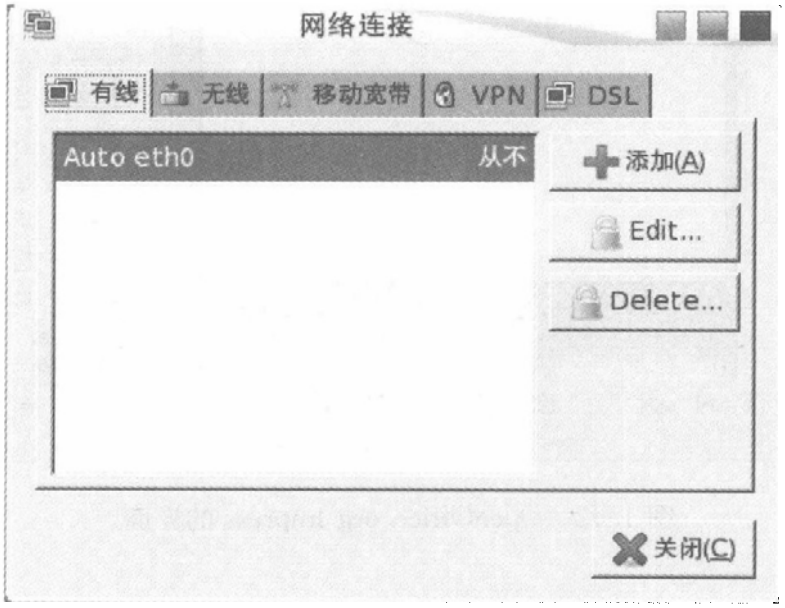


图 1.43 网络连接对话框图

从图 1.43 中可以看到当前系统已经有了一个默认的网络配置“Auto eth0”，可以选中它单击“Edit”按钮对其进行编辑，也可以单击“添加”按钮来安装配置一个新的设备。本例将新建一个网络连接，任何对其进行配置。单击“添加”按钮，打开如图 1.44 所示的“正在编辑 有线连接 1”对话框。在“IPv4 设置”选项卡中对网卡 IP 地址进行配置，在方法下拉框中有 5 种方式，用户可以通过 DHCP 服务自动获取 IP 地址，也可以采取手动配置静态 IP 地址。本例采用的是后者，单击“添加”按钮，在“地址”栏分别输入 IP 地址、子网掩码以及默认网关，在“DNS 服务器”栏输入 DNS 服务器的名称或 IP 地址，这些信息来自局域网的网络管理中心。

点击“应用”按钮，保存刚刚修改的配置。

至此配置完毕，可以打开浏览器进行浏览了。

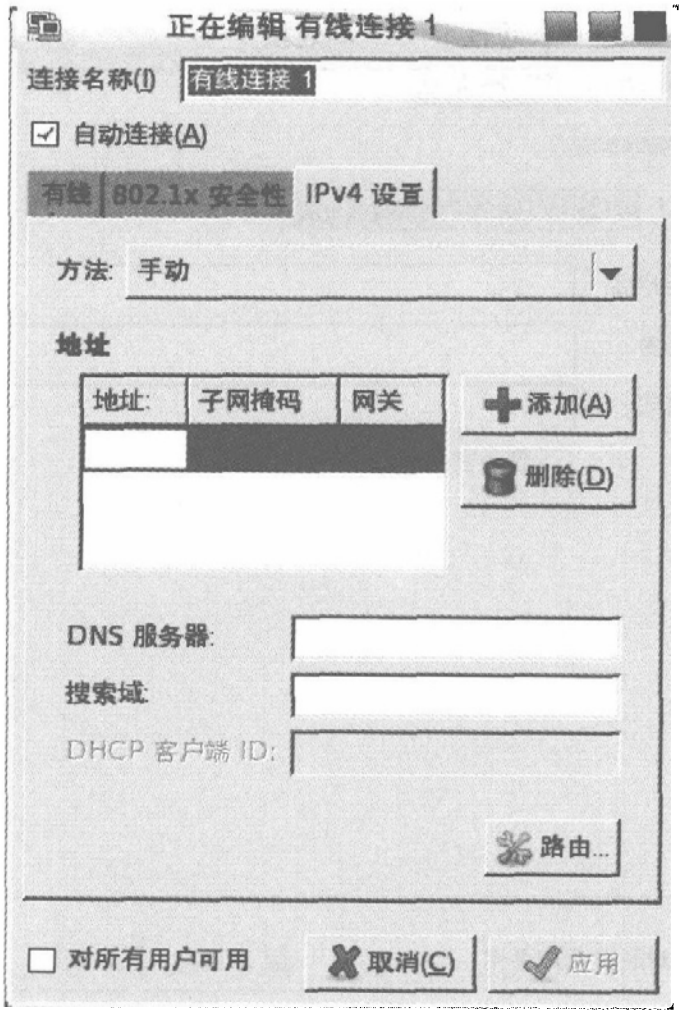


图 1.44 IP 地址设置对话框

2. 建立 DSL 连接

在图 1.43 所示的“网络连接”对话框中选中最右边的“DSL”，单击“添加”按钮，出现“正在编辑 DSL 连接 1”对话框，如图 1.45 所示。

选中“DSL”选项卡，在“用户名”、“服务”和“密码”的文本框中输入你的 DSL 的用户名、DSL 服务提供商的服务(可输入任意字符)，以及密码(DSL 服务提供商提供)。单击“应用”按钮结束配置。

至此，配置完毕，可以打开浏览器进行浏览。

如果想断网，则右击 KDE 面板右端的  图标，单击“启用联网”去掉前面的钩号即可。

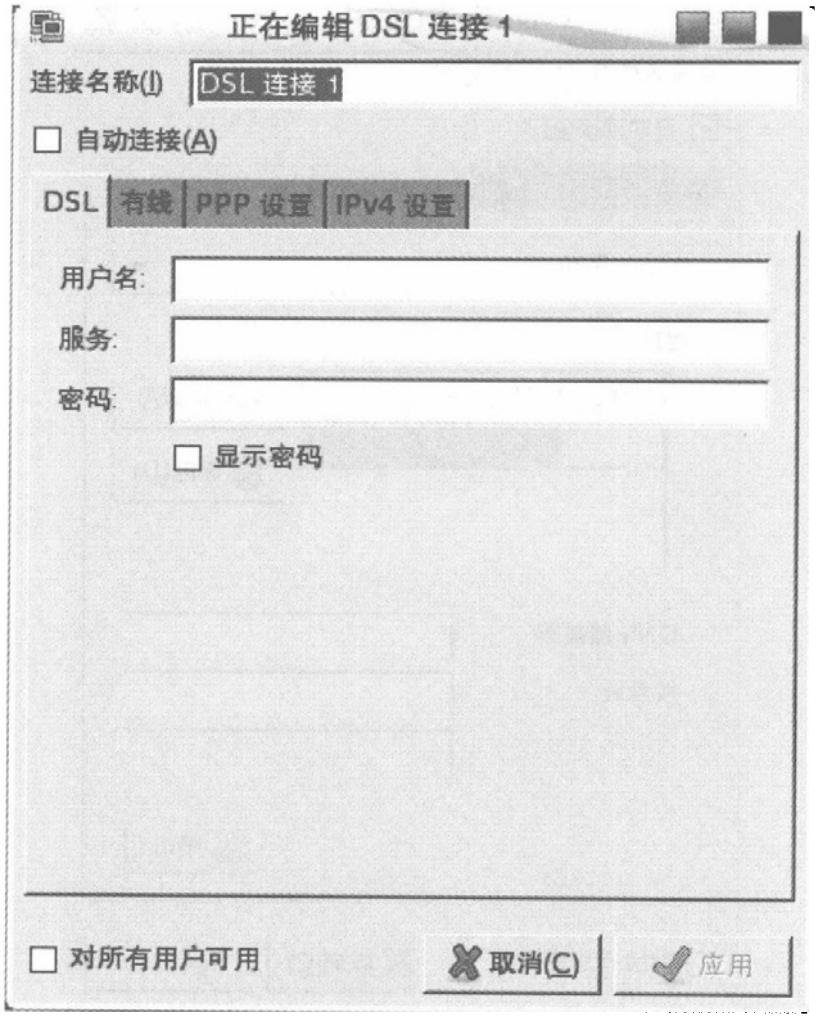


图 1.45 正在编辑 DSL 连接 1 对话框

1.7.2 Web 浏览

Mozilla 是 Red Flag Linux 中的一个浏览器软件，执行“应用程序”→“Internet”→“Firefox Web Browser”命令或单击面板上 Mozilla 快速启动按钮即可启动 Mozilla，启动后的界面如图 1.46 所示。

Mozilla 的功能和其他的浏览器一样，它也具有标准的导航工具栏、按钮和菜单等。在导航栏中的地址栏中输入要访问的网页地址，如：http://www.baidu.com，点击“转到”按钮即可打开对应的网址进行浏览。

Mozilla 还支持关键字搜索，在“转到”按钮后的搜索栏中输入要搜索的关键字，如“Linux”，按下 Enter 键，即可使用指定的搜索引擎进行搜索。指定搜索引擎可以通过点击搜索栏中的图标打开搜索引擎下拉列表框进行选择。

图 1.46 Red Flag Linux 中 Mozilla 浏览器的默认界面

2 Scilab 基础

2.1 Scilab 概述

Scilab(Scientific Laboratory) 是以法国国立信息与自动化研究院(INRIA)的科学家为主共同开发的"开放源码"式科学计算软件,它主要有两个功能:数值计算和计算结果可视化。Scilab 数据类型丰富,可以很方便地实现各种矩阵运算。Scilab 也能处理比数字矩阵复杂得多的对象,如控制专业的多项式传递函数矩阵。Scilab 允许用户在线建立自定义函数。函数在 Scilab 中被当作数据对象处理。另外,Scilab 具有功能丰富的图形显示能力,可以完成各种常规形式计算结果的可视化。

Scilab 为用户提供如下计算和开放式编程环境:

(1) 多种容易操作的数据类型。

(2) 一个作为广泛计算基础的合理有效的基本函数集。

(3) 一个开放式编程环境,新的函数能很容易地被添加。

Intersci 是一个有用的发布工具,通过它能建立接口,添加新的函数及工具箱,例如,增加新的 FORTRAN 代码和 C 代码到 Scilab 中。Scilab 还包括一些应用于不同科学计算领域的工具箱,例如,应用于数学建模、信号处理、网络分析、决策优化、线性与非线性控制等多个方面的工具箱。它的工具箱允许图形定义和模拟复杂的连续和离散的混杂系统。Scilab 由三个独立的部分组成:解释器、函数库(Scilab 程序)以及 FORTRAN 和 C 程序库。另外,Scilab 是一种解释性语言,能运行于 Windows、Linux 以及 Unix 等操作系统环境下。Scilab 与目前流行的 Matlab 软件起源相同,都源自 Cleve Moler 于 1980 年开发的程序,其功能与 Matlab 软件相似,并且表达式的语法、函数的调用和大多数控制指令都相似。

2.2 Red Flag Linux 下 Scilab 的运行环境与安装

2.2.1 下载 Scilab 的 Linux 版本

要安装 Scilab,首先访问它的下载页面(图 2.1):http://www.scilab.org/

products/scilab/download/，下载 Scilab 5.2.2 的 Linux(同时适用于 32 位和 64 位)版本，用户可以获得一个二进制文件的版本：scilab-5.2.2.bin.linux-i686.tar.gz。

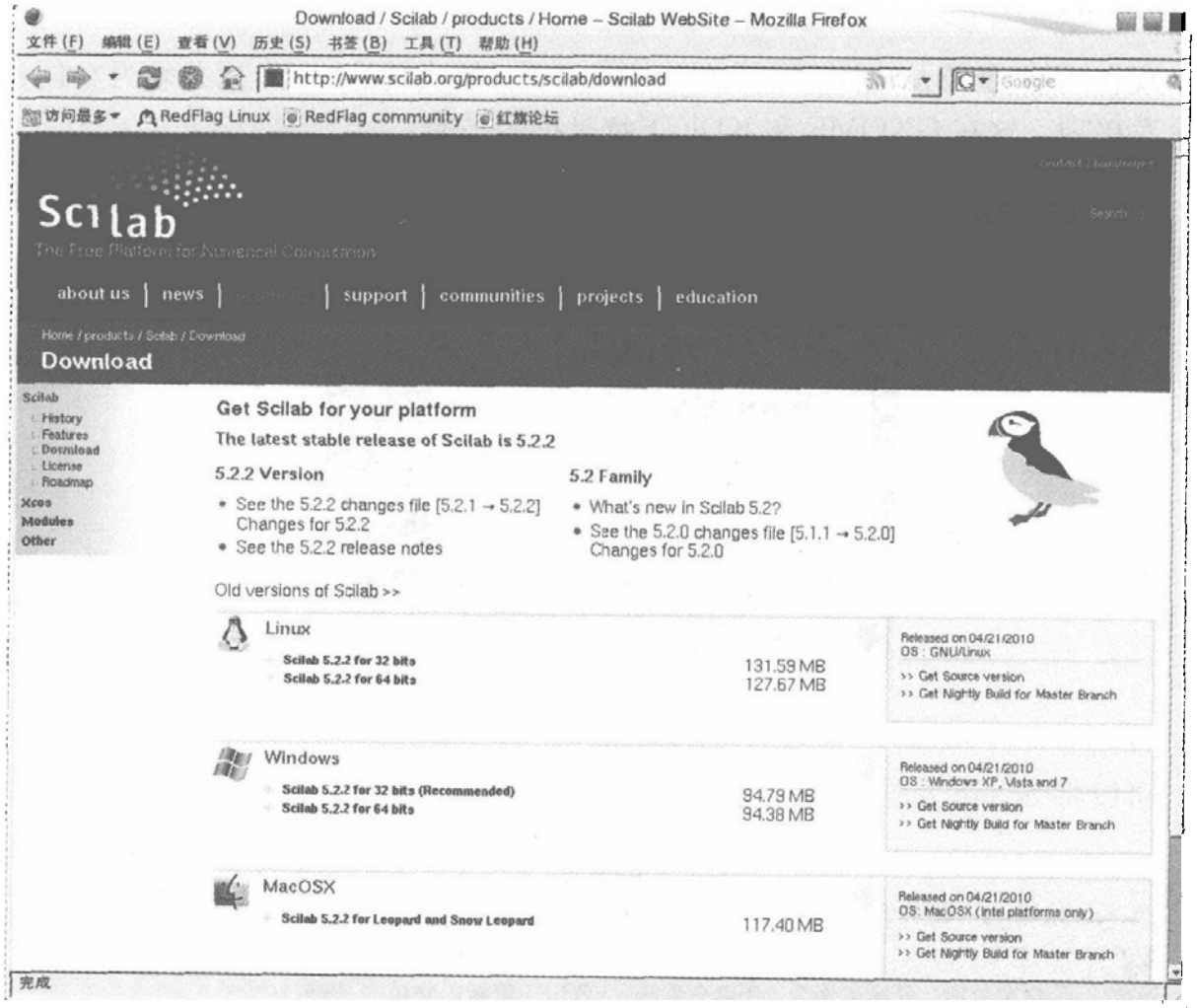


图 2.1 Scilab 5.2.2 下载页面

2.2.2 解压文件

当下载完成后，可将文件保存到桌面，解压文件(右击该文件并选择解压到当前目录)或通过终端(应用程序/附件/终端)执行如下命令：

```
cd $ HOME/桌面/
tar -xzvf scilab-5.2.2.bin.linux-i686.tar.gz
```

现在可以从解压的目录中运行 scilab，只要进入/scilab-5.2.2/bin 目录，双击名称为 scilab 的文件就可以打开了。但这并不是最理想的做法。做一个更完整的安装，可以移动 scilab 到用户的目录/home 下面，或者更好的是移动到用户根目录下面的/opt(这是指为第三方应用的目录)，以下方法是基于后面一种情况：

由于开始解压的 scilab 包仍保存在桌面上，需要将其移到/opt 目录。打开终

端，执行以下命令：

```
mv  scilab-5.2.2/  /opt/
```

2.2.3 创建 Scilab 菜单项

现在，需要能够从应用程序菜单启动 scilab。要实现这一点，就必须为它创建一个菜单项。这在 GNOME 和 KDE 下都是很容易的。

要在用户的 KDE 面板下创建一个 Scilab 应用程序菜单项，右击 并选择菜单编辑器，如图 2.2 所示。

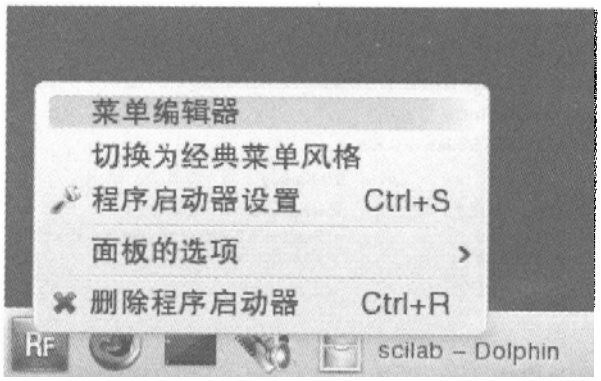


图 2.2 启动“菜单编辑器”面板

可以将 Scilab 安装在名为“教育”的菜单下面，如图 2.3 所示，右击教育（在菜

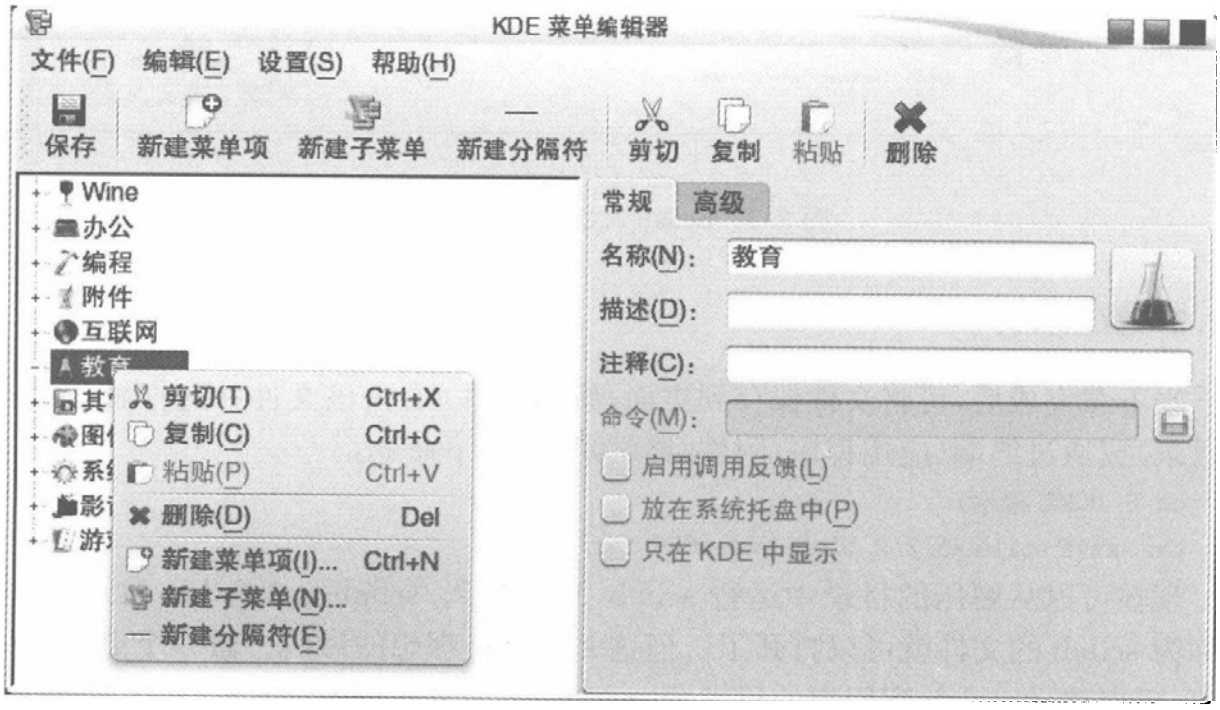


图 2.3 新建 Scilab 菜单项

单编辑器的左侧),选择“新建菜单项”(在菜单编辑器的右侧)在教育菜单下面创建新的菜单项。这样做会打开一个新的对话框,然后按照下面的内容进行填写(图 2.4、图 2.5):

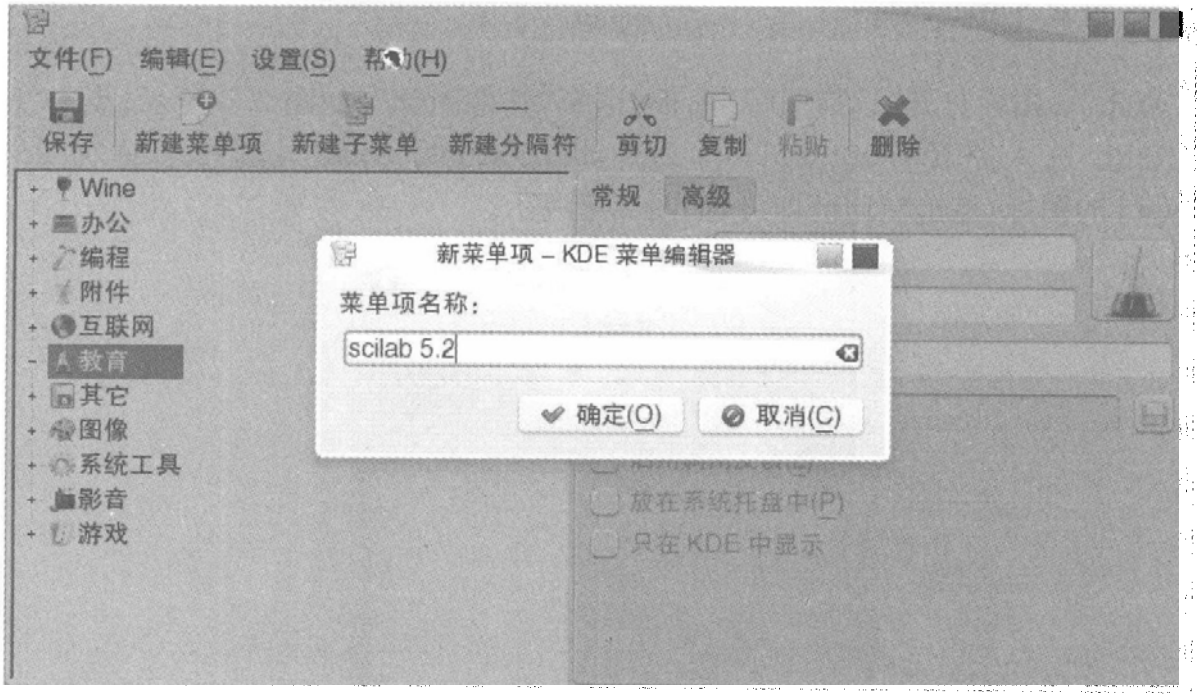


图 2.4 设置菜单项名称

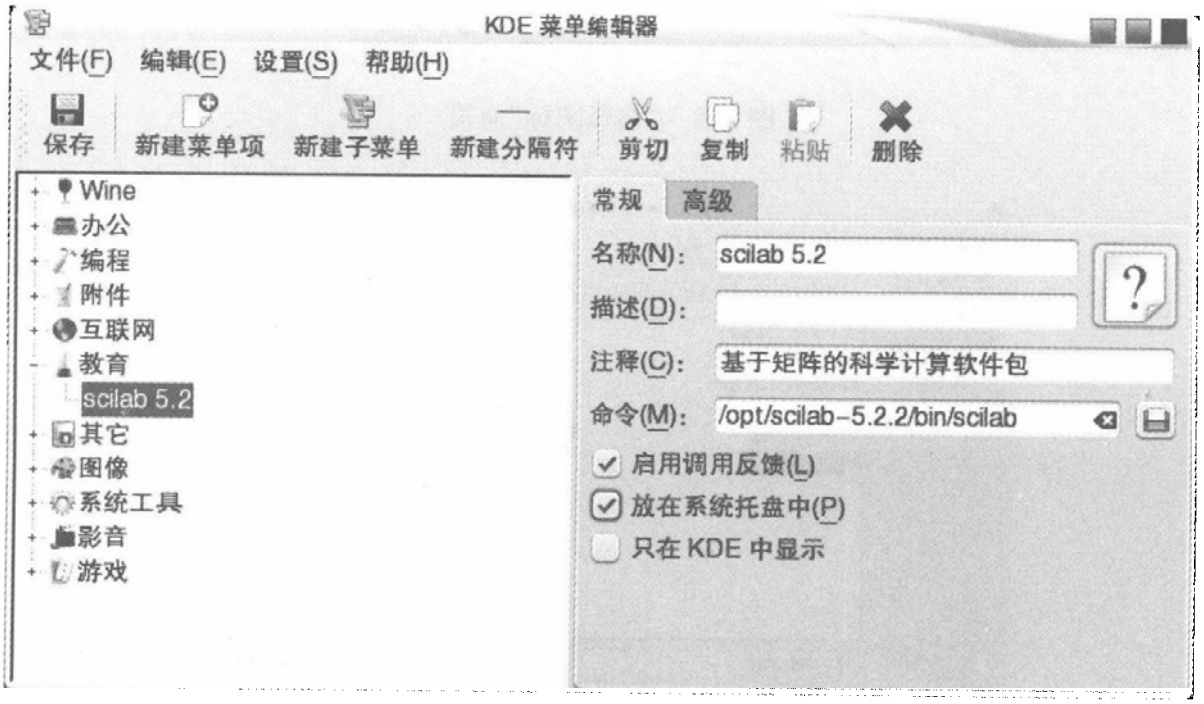


图 2.5 Scilab 菜单基本信息设置

名称：Scilab-5.2

注释：基于矩阵的科学计算软件包

命令：/opt/scilab-5.2.2/bin/scilab

2.2.4 设置 Scilab 图标

点击 图标打开“选择图标”面板，选择“其他图标”，如图 2.6 所示；点击“浏览”按钮，进入路径“/opt/scilab-5.2.2/share/scilab/icons/”，选择名称为 scilab.xpm 的图标，点击“打开”即可，如图 2.7 所示。

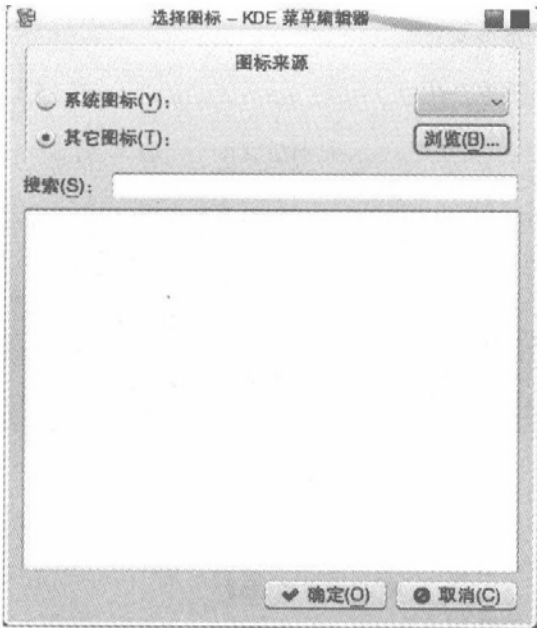


图 2.6 “选择图标”面板

图 2.7 “图标设置”面板

现在就可以从系统菜单/教育/Scilab/下面启动 Scilab 了，如图 2.8、图 2.9 所示。

图 2.8 “教育”菜单

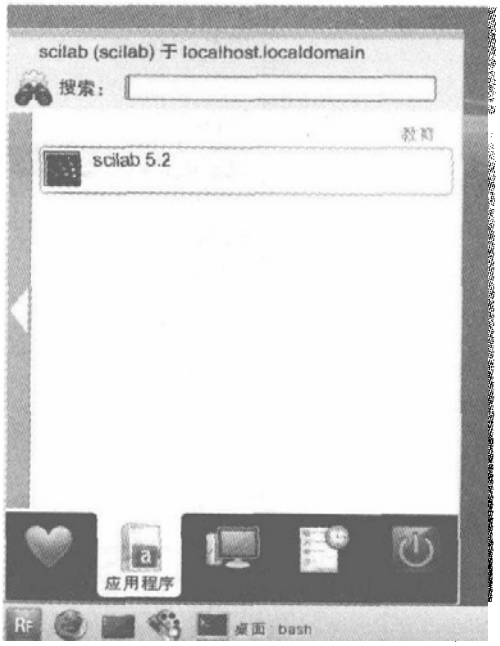


图 2.9 Scilab 启动项

2.2.5 解决不支持中文显示的问题

如果 Scilab 主窗口中显示乱码(不支持中文)，如图 2.10 所示，点击 A 按钮，将字体设置成中文字体格式(如“东文宋体”)即可，如图 2.11 所示。

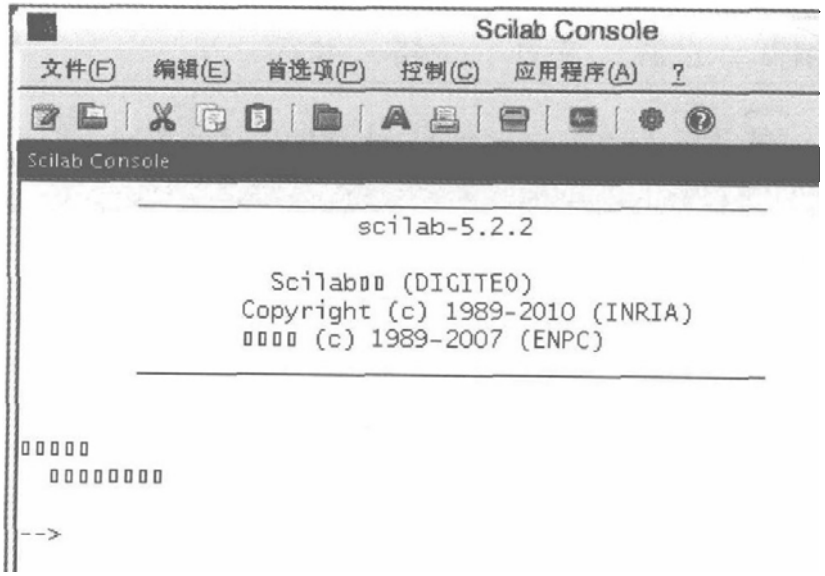


图 2.10 主窗口中显示乱码

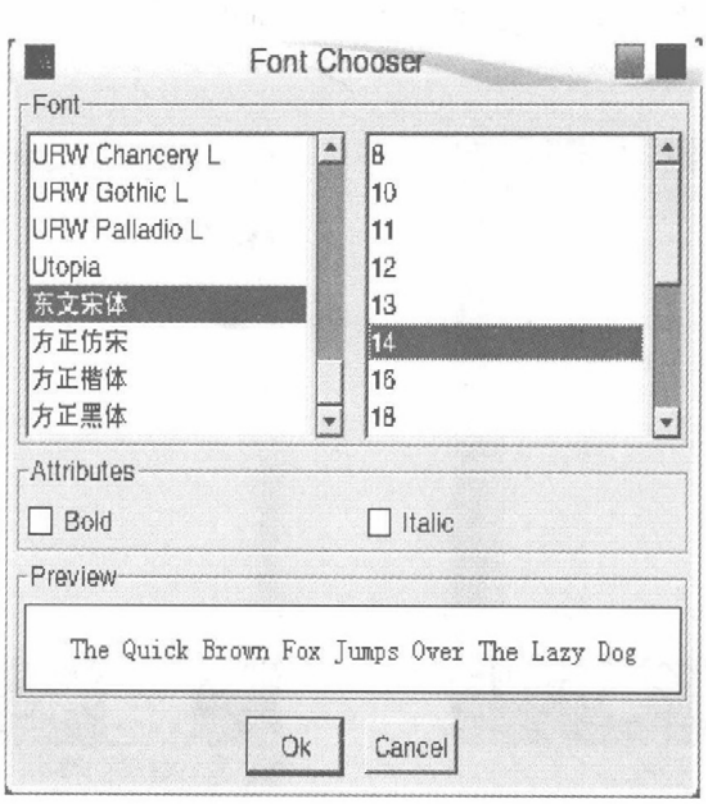


图 2.11 “字体设置”面板

如果 Scilab 标题栏或菜单栏中显示乱码(不支持中文),如图 2.12 中黑色标题栏前面的方格所示。这是由于 Scilab 的 Java 没有中文字体库,解决办法就是将系统中的中文字体复制到 Scilab 的 Java 字体目录下面,具体过程如下:

图 2.12 标题栏中显示乱码

(1) 打开终端,在其中执行命令:

```
cd /opt/scilab-5.2.2/thirdparty/java/lib/fonts
```

进入到 Scilab 的 Java 字体目录;

(2) 在 Scilab 的 Java 字体目录下创建名称为 fallback 的目录：

```
mkdir fallback
```

(3) 进入创建的目录：

```
cd fallback
```

(4) 将 Red Flag Linux 桌面 7.0 下的中文字体文件链接到此处：

```
ln /usr/share/fonts/zh_CN/TrueType/fzht.ttf
```

此处使用的是方正黑体，用户可以根据自身的需要使用其他格式的字体，字体文件是/usr/share/fonts/zh_CN/TrueType/目录下的扩展名为 .ttf 的文件。

此时只需要重新启动 Scilab，就可以看到原先的方格已经能够正常显示中文字体了，如图 2.13 所示。

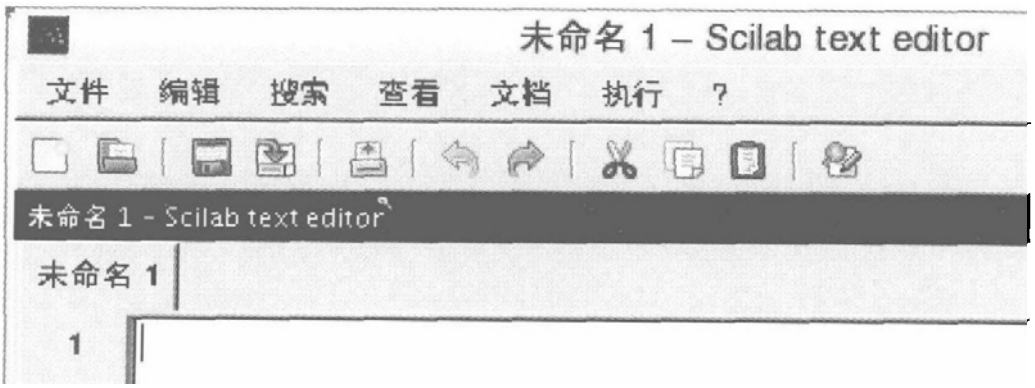


图 2.13 乱码修复后的标题栏

如果在 Scilab 程序运行的过程中的中文显示出现乱码，例如，程序中的信息框中不显示中文，解决办法是在“Scilab text editor”的菜单栏中执行“文档”→“Encoding”，选择字体编码格式为“UTF-8”即可。

2.3 Scilab 集成环境

Scilab 主界面如图 2.14 所示。

“文件”菜单：

->Exec 执行一个 Scilab 程序文件

->Open 打开一个 Scilab 程序文件

->Load 加载 Scilab 数据文件或者编译过的函数文件

->Save 保存 Scilab 变量数据文件

->Change Directory 改变当前的 Scilab 工作区目录

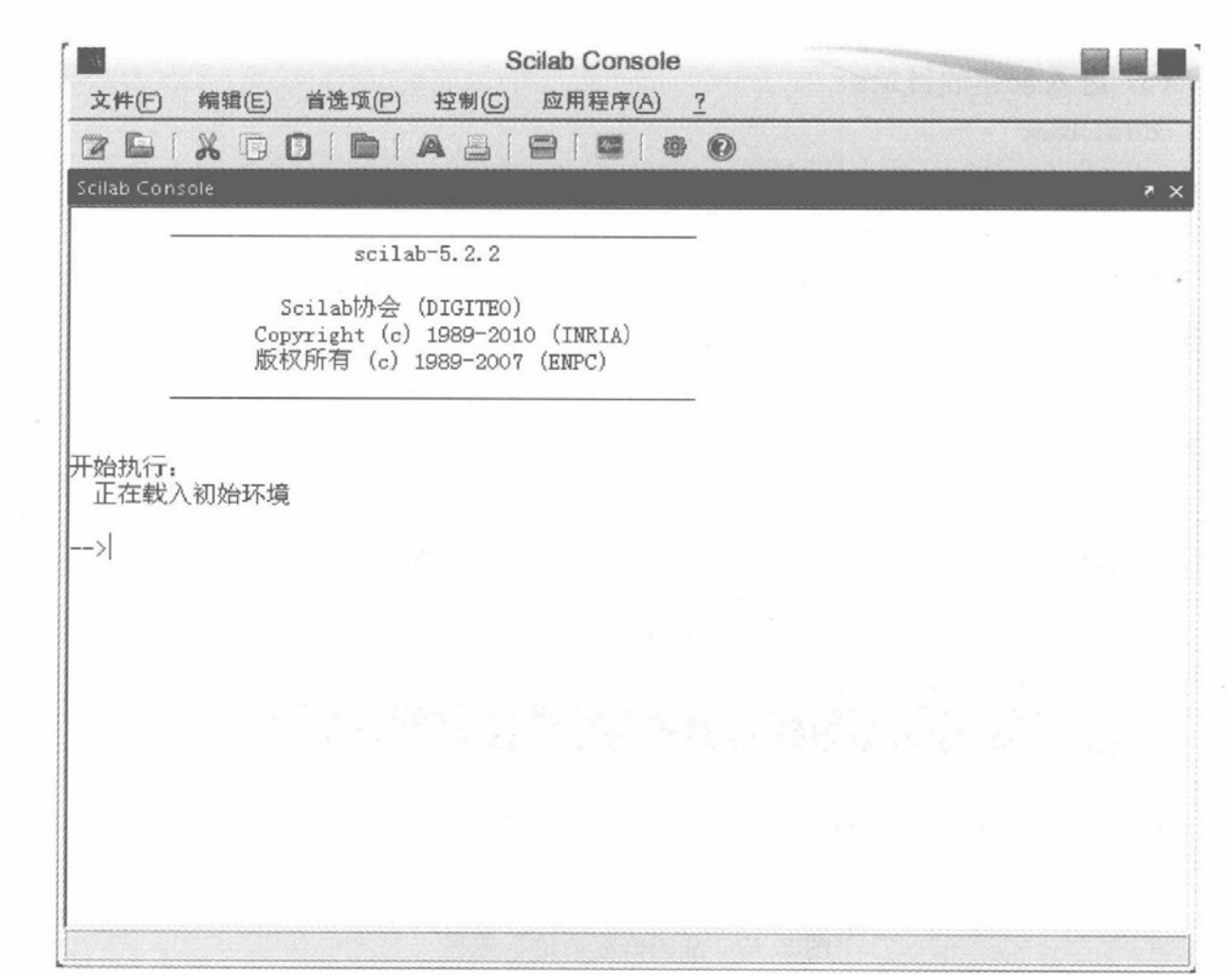


图 2.14 Scilab 主界面

->Get Current Directory 得到当前的 Scilab 工作区目录

->Print Setup 打印设置

->Print 打印

->Exit 退出

“编辑”菜单:

->Select all 选择目前工作区所有文本

->Copy 复制文本

->Paste 粘贴

->Empty Clipboard 清空剪贴板

“首选项”菜单:

->Language 选择语言

->Colors 用户可以自己定义界面的显示颜色

->Toolbar 显示和隐藏工具栏

->File's Association 设置文件关联

->Choose Font 设置字体

->Clear History 按键盘的上下键可以选择最近输入的 Scilab 命令，本选项清除最近输入的命令提示

->Clear Command Window 清空屏幕显示

->Console 打开 Scilab 命令行形式

“控制”菜单：

->Resume 回复

->Abort 取消运行

->Interrupt 中止运行

Editor 菜单

打开 Scilab 程序编辑器

Applications 菜单：

->Scicos Scilab 图形化仿真模块

->Edit Graph 编辑图形

->M2SCI 将 Matlab 语言的命令程序转换为 Scilab 的程序

->Browser Variables 察看当前工作区的各个变量的值，并且可以编辑

? 菜单

->Scilab HELP 帮助

->Configure 帮助的显示方式设置

->Scilab Demos Scilab 功能演示

2.4 Scilab 帮助系统

Scilab 具有详尽的帮助文档，并且具有查找功能，可以帮助用户快速找到所需要的内容，如图 2.15 所示。

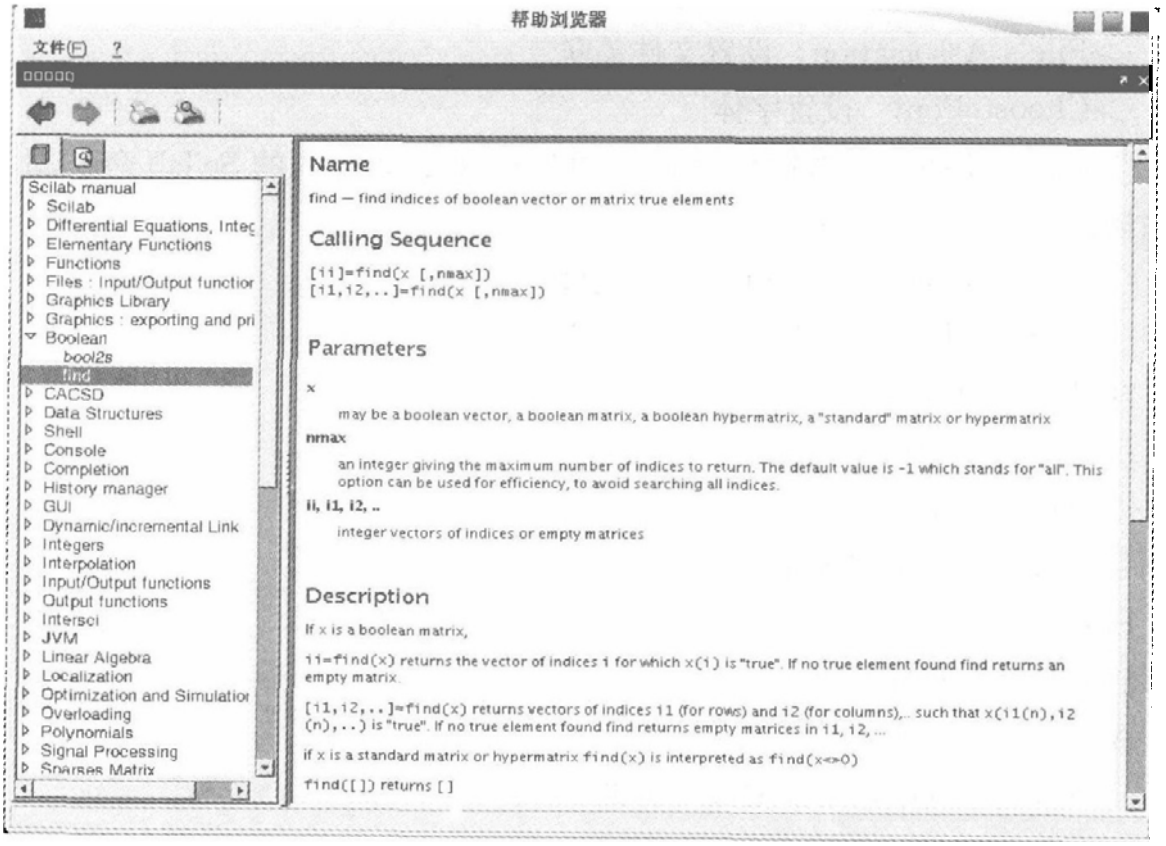


图 2.15　Scilab 帮助浏览器界面

3 Scilab 数据

Scilab 具有丰富的数据类型，如整型、字符型、布尔型等，以上的数据类型都以矩阵的形式存在，所以矩阵是 Scilab 最基本的数据对象。本章重点介绍 Scilab 中各种数据的表示方法和基本运算。

数据作为计算机处理的对象，在程序语言中可分为多种类型，Scilab 作为一种可编程的语言当然也不例外。Scilab 的主要数据类型如下所示：

(1) 符号与变量；
(2) 数值型常量；
(3) 数值型向量与矩阵；
(4) 整型数组；
(5) 布尔型向量与矩阵；
(6) 字符串型数据；
(7) 多项式类型；
(8) 表类型。

3.1 Scilab 数据的特点

矩阵是 Scilab 最基本的数据对象，Scilab 的大部分运算是在矩阵运算的意义下执行的。Scilab 下的变量不需要预先定义，其直接根据第一次使用时的数据类型自动确定类型，这在高级语言中是极有特色的。这样处理的好处是，在书写程序时可以随时引入新的变量而不用担心会出什么问题，这的确给应用带来了很大方便。但缺点是有失严谨，会给搜索和确定一个符号是否为变量名带来更多的时间开销。

3.2 变量和数据操作

3.2.1 变量命名

Scilab 中的变量命名，变量以字母开头，后接字母、数字、下划线。Scilab 中变量区分大小写。Scilab 中函数调用时必须使用小写，否则就会出错，在使用 Scilab 语言编程时一定要注意进行区分。例如，在 Scilab 中，AB、Ab 与 ab 均是不同的

变量。

3.2.2 变量赋值

Scilab 中变量可以直接赋值，也可以使用已知变量组成的表达式赋值。如：

a=1

b=a+1

Scilab 命令语句后面可以不使用结束符号，但如果使用“;”则表示不在屏幕输出此项运算显示。例如，输入以下命令：

```
-->a = 1.17 * 5,  b = 1.29 + 1.07
```

则屏幕上显示

```
a  =

  5.85
b  =

  2.36
```

如将两个表达式中间的“,”改成“;”，即

```
-->a = 1.17 * 5;  b = 1.29 + 1.07
```

则屏幕显示如下：

```
b  =

  2.36
```

Scilab 中的注释采用符号“//”并且只支持单行注释。即 Scilab 不执行“//”(包括“//”在内)以后的内容。注释行只供用户阅读，用以说明某段程序代码的作用、某个语句的作用或作者的编程思想等。例如，输入以下命令：

```
-->c = 1.17   // d = 1.29
```

屏幕只显示

```
c  =

  1.17
```

其中，d=1.29 并没有执行，注意这与上面执行的命令 a=1.17 * 5;b=1.29+1.07 有本质区别。

命令 a=1.17 * 5; b=1.29+1.07 执行了两条命令，但执行结果 a=5.85 是由于后面加了分号，没有显示出来，但此时 a 这个变量已经保留在 Scilab 工作区中，用户直接输入 a，如下：

```
-->a
```

屏幕上就会显示

```
a  =

    5.85
```

而命令 c=1.17 // d=1.29 实质上只执行了 c=1.17，而后面的部分 d=1.29 没有执行，如果用户直接输入 d，如下：

```
-->d
```

系统会给出如下的错误提示：

```
 ! --error 4
未定义的变量:d
```

这说明 Scilab 工作区中根本就没有变量 d。

用户可以使用 clear 命令用来清除工作区中没有被 predef 命令保护的所有变量内容；也可使用"clear 变量名"的格式来清除工作区中没有被 predef 命令保护的指定变量。通常情况下，受保护的变量是指 Scilab 标准库定义的变量和使用%前缀定义的变量。用户可以使用 clear 和 predef(0)两条命令来清除工作区中所有的变量。

注意清除变量 a 的语法是 clear a 而不是 clear(a)，a=[]并没有清除变量 a，而是将一个空矩阵赋值给了 a。

3.2.3 特殊变量和常数

ans：当函数没有指定的输出时，结果就放在变量 ans 中。

%pi：圆周率 π。

%i：虚数单位。

%e：自然对数的底。

%eps：用户的计算机所能识别的最小的数，即浮点运算的相对精度，可以用来表示 Scilab 计算的误差。

%inf：无穷大，可以用来避免浮点溢出。使用 isinf(a) 可以来判别 a 是否是无穷大。

%nan：不是一个数或不能确定的数(Not a Number)，可以表示一个无效的数值。使用 isnan(a) 可以来判别 a 是否是"Not a Number"类型。

%T，%t，%F，%f：布尔常量(真假值)。

%io：二维常向量。

3.2.4 内存变量管理

可以使用 Browser Variables 菜单或者 browsevar()；命令来在变量浏览器中

察看 Scilab 中预定义的各个常数。

如图 3.1 所示，打开变量浏览器，设置显示预定义变量。

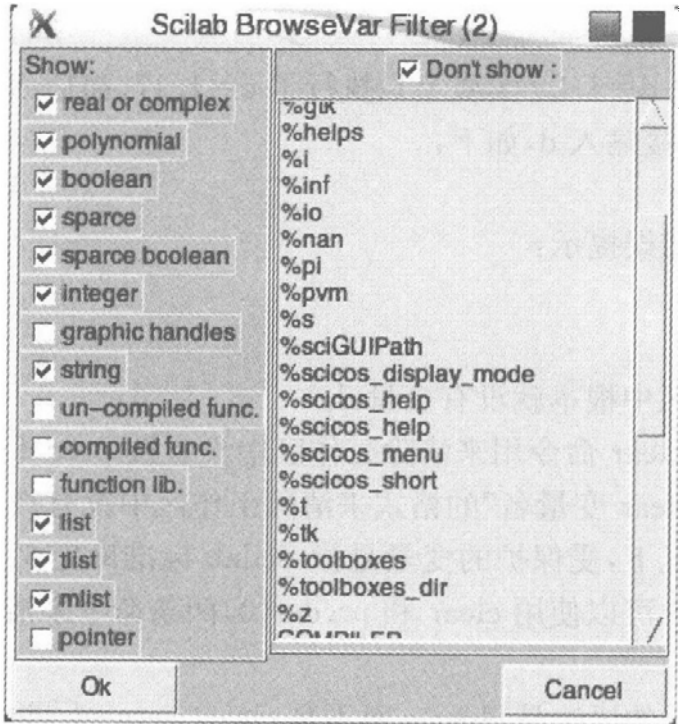


图 3.1 Scilab 变量浏览器

Scilab 的内存变量管理使用 Browser Variables 完成。可以完成变量的查看、编辑与删除，如图 3.2 所示。Scilab 工作区中的变量可以通过 Save 和 Load 命令进行保存和读取。

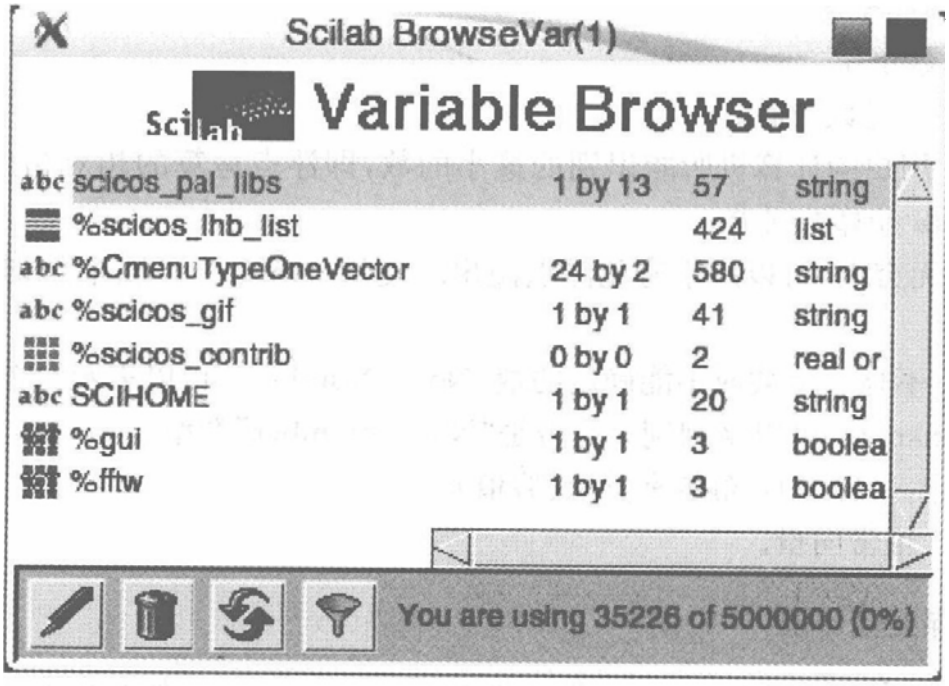


图 3.2 Scilab 内存变量管理

命令如下：

Save“文件路径”

Load“文件路径”

3.3 Scilab矩阵

3.3.1 标量的生成

标量的生成很简单，只需要直接给出一个变量赋值：

```
-->a = 1
a  =


    1.
```

3.3.2 向量的生成

向量的生成主要有3种方法：直接输入数据、利用冒号表达式生成。

1. 直接输入数据

直接输入数据时，行向量的各分量间用空格或者逗号隔开，列向量的分量间用分号隔开。

直接赋值产生向量。

```
-->s = [1 2 3 4]          //生成一个行向量，之间采用空格
s  =
    1.    2.    3.    4.
-->s = [1; 2; 3; 4]       //生成一个列向量，之间采用；

s  =


    1.
    2.
    3.
    4
```

2. 利用冒号表达式

利用“：”可以自动生成均匀等份的向量。其格式为y=a：step：b，式中a为向量的第一个元素，b为向量最后一个元素的限定值，step是变化步长，省略步长

时系统默认为 1。step 不一定为整数，当 step 为正数时，a 必须小于 b，否则生成空向量；反之亦然。

利用“：”生成向量：

```
-->a = 1 : 5          //省略步长时默认为 1
 a  =

    1.    2.    3.    4.    5.
-->b = 1 : 2 : 10     //指定步长生成向量
 b  =

    1.    3.    5.    7.    9.
```

这种给向量赋值的方式在编程中比较常见。

```
-->c = 10 : -1 : 1   //步长为负数时生成递减的向量
 c  =


          column 1 to 8

   10.    9.    8.    7.    6.    5.    4.    3.

          column  9 to 10

    2.    1.
-->d = 10 : 2 : 4     //当第一个元素大于最后一个元素，步长为正数时生成空向量
 d  =

    []
-->e = 2 : -1 : 10   //当第一个元素小于最后一个元素，步长为负数时生成空向量
 e  =

    []
```

3. 利用函数生成

利用 linspace 函数生成元素个数可以明确给定的等份向量，其调用格式为

[y]=linspace(d1,d2,n)

函数返回在 d1 与 d2 之间均匀分布的有 n 个元素值的行向量，n 的缺省值

是 100。

利用 linspace 函数生成向量：

```
-->u = linspace(1,100,10)
 u  =

        column 1 to 7

   1.    12.    23.    34.    45.    56.    67.

        column  8 to 10

   78.    89.    100.
```

但 n 缺省时，[y]=linspace(d1,d2)在 d1 与 d2 之间自动生成有 100 个元素的线性等分向量。

如果需要对数等分向量，即元素的对数值线性等分的向量，Scilab 提供 logspace 函数实现这一功能。函数调用格式为

[y]=logspace(d1,d2,n)

函数返回在 10^{d1} 与 10^{d2} 之间的 n 个元素构成的向量，这些元素以对数基为线性等分。n 的缺省值是 50。

利用 logspace 函数生成向量：

```
-->v = logspace(1,3,3)
 v  =

   10.    100.    1000.
```

注意：

(1) b 在冒号表达式中，它不一定恰好是向量的最后一个元素，只有当向量的倒数第二个元素加步长等于 b 时，b 才正好构成尾元素。如果一定要构成一个以 b 为末尾元素的向量，那么最可靠的生成方法是用线性等分函数。如下面的例子所示：

```
-->s = 1 : 0.5 : 2.9
 s  =

   1.    1.5    2.    2.5

-->v = linspace(1,2.9,4)
 v  =
```

```
    1.    1.6333333    2.2666667    2.9
```

(2) 在使用线性等分函数前，必须先确定生成向量的元素个数 n，但使用冒号表达式将依着步长和 b 的限制去生成向量，用不着去考虑元素个数的多少。

(3) 实际应用时，同时限定尾元素和步长去生成向量，有时可能会出现矛盾，此时必须做出取舍。要么坚持步长优先，调整尾元素限制；要么坚持尾元素限制，去修改等分步长。

3.3.3 矩阵的生成

在 Scilab 中建立矩阵的方法很多，本节主要介绍 5 种，分别是直接输入法、抽取法、拼接法、函数法和加载法。不同的方法往往适用于不同的场合和需要。

因为矩阵是 Scilab 特别引入的量，所以在表达时，必须给出一些相关的约定与其他量区别，这些约定是：

矩阵的所有元素必须放在方括号[]内；每行的元素之间需用逗号或空格隔开；矩阵的行与行之间用分号或回车符分隔；元素可以是数值或表达式。

1. 直接输入法

简单矩阵生成的常用方式是直接从键盘输入矩阵。

生成一个 3×4 的矩阵：

```
-->m=[1,2,3,5;4,5,2,3;5,3,1,9]
 m  =

    1.    2.    3.    5.
    4.    5.    2.    3.
    5.    3.    1.    9.
```

分行输入：

```
-->m=[1 2 3 5
-->4 5 2 3
-->5 3 1 9]
 m  =

    1.    2.    3.    5.
    4.    5.    2.    3.
    5.    3.    1.    9.
```

在以上两个输入形式中：第一种形式的行内元素用逗号隔开，行之间用分号隔开；第二种形式的行内元素用空格隔开，行之间用回车符隔开，但所赋的数值内容是一样的。已生成的数据保存在 Scilab 工作空间中，可以通过调用变量名“m”进

行使用，如显示、参与运算等。

2. 抽取法

抽取法是从大矩阵中抽取出需要的小矩阵(或子矩阵)。线性代数中分块矩阵就是一个典型的从大矩阵中取出子矩阵块的实例。矩阵的抽取实质是元素的抽取，用元素下标的向量表示从大矩阵中去提取元素就能完成抽取过程。

```
-->A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]
 A  =

    1.    2.    3.    4.
    5.    6.    7.    8.
    9.    10.   11.   12.
    13.   14.   15.   16.

-->B=A(1:3,2:3)          //取矩阵A行数为1～3,列数为2～3的元素构成子矩阵B
 B  =

    2.    3.
    6.    7.
    10.   11.

-->C=A([1 3],[2 4])      //取矩阵A行数为1、3,列数为2、4的元素构成子矩阵C
 C  =

    2.    4.
    10.   12.

-->D=A(4,:)              //取矩阵A第4行,所有列,":"可表示所有行或列
 D  =

    13.   14.   15.   16.
```

3. 拼接法

行数与行数相同的小矩阵可在列方向扩展拼接成更大的矩阵。同理，列数与列数相同的小矩阵可在行方向扩展拼接成更大的矩阵。

```
-->A=[1 2 3;4 5 6;7 8 9],B=[9 5;7 6;2 8],C=[1 5 0;7 1 3]
 A  =
```

```
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
 B  =

    9.    5.
    7.    6.
    2.    8.
 C  =

    1.    5.    0.
    7.    1.    3.

-->M=[A B;B A]        //行列两个方向同时拼接,请留意行、列数的匹配问题
 M  =

    1.    2.    3.    9.    5.
    4.    5.    6.    7.    6.
    7.    8.    9.    2.    8.
    9.    5.    1.    2.    3.
    7.    6.    4.    5.    6.
    2.    8.    7.    8.    9.

-->N=[A;C]        //A、C 列数相同,沿行向扩展拼接
 N  =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
    1.    5.    0.
    7.    1.    3.
```

4. 函数法

常用矩阵的生成函数：

全 0 阵，即元素全为 0 的矩阵。函数名为 zeros，调用格式如下：

zeros(m1,m2)：生成 m1×m2 的全 0 矩阵。

zeros(m1,m2,…,mn)：生成 m1×m2×…×mn 的全 0 矩阵。

zeros(A):生成大小与矩阵 A 相同的全 0 阵。

zeros():返回单个 0 值,相当于 zeros(1,1)。

全 1 阵,即元素全为 1 的矩阵。函数名为 ones,调用格式如下:

ones(m1,m2):生成 m1×m2 的全 1 矩阵。

ones(m1,m2,…,mn):生成 m1×m2×…×mn 的全 1 矩阵。

ones(A):生成大小与矩阵 A 相同的全 1 阵。

ones():返回单个 1 值,相当于 ones(1,1)。

单位阵,即对角线元素全为 1,其他元素为 0 的矩阵。函数名为 eye,调用格式如下:

eye(m,n):生成 m×n 的对角线元素全为 1,其他元素为 0 的矩阵。

eye(A):生成大小与矩阵 A 相同的单位阵。

eye():返回单个 1 值。

均匀分布随机阵,即各个元素为 0～1 间均匀分布的随机数的矩阵。函数名为 rand,调用格式如下:

rand(m1,m2):生成 m1×m2 的随机阵。

rand(m1,m2,…,mn):生成 m1×m2×…×mn 的随机阵。

rand(A):生成大小与矩阵 A 相同的随机阵。如果 A 是个复数阵,那么 rand(A)也是个复数阵。

rand():返回一个随机标量。

常用矩阵的生成如下:

```
  //zeros: 生成全 0 矩阵
-->zeros(3,2)
 ans    =

    0.    0.
    0.    0.
    0.    0.

-->zeros(3,2,3)
 ans    =

(:,:,1)

    0.    0.
```

```
    0.    0.
    0.    0.
(:,:,2)

    0.    0.
    0.    0.
    0.    0.
(:,:,3)

    0.    0.
    0.    0.
    0.    0.

//ones:生成全1矩阵
-->s = ones(3,2)
 s  =

    1.    1.
    1.    1.
    1.    1.

//eye:单位矩阵
-->s = eye(2,2)
 s  =

    1.    0.
    0.    1.


//rand:随机矩阵
-->s = rand(3,3)
 s  =

    0.2113249    0.3303271    0.8497452
    0.7560439    0.6653811    0.6857310
    0.0002211    0.6283918    0.8782165
```

```
//空矩阵:生成没有任何元素的空矩阵

  -->s = []
s   =

    []
```

5. 加载法

加载法是指将已经存放在外存中的 .dat 文件读入 Scilab 工作空间中。这一方法的前提是必须在外存中事先已保存了该 .dat 文件且数据文件中的内容是所需的矩阵。

在用 Scilab 编程解决实际问题时,可能需要将程序运行的中间结果用 .dat 保存在外存中以备后面的程序调用。这一调用过程实质就是将外存中的数据(包括矩阵)加载到 Scilab 内存工作空间以备当前程序使用。

加载的方法具体有菜单法和命令法,在命令窗口中交互讨论问题时,用菜单和用命令都可用来加载数据,但在程序设计时就只能用命令去书写程序了。具体来说,加载用的菜单是命令窗口中的文件→装载运行环境,而命令则是 load。

```
-->a = eye(2,2);b = ones(a);

-->save('vals.dat',a,b);   //保存 a,b

-->clear a   //清除当前工作空间中的变量 a

-->clear b

-->a   //当前工作空间中的变量 a 已被清除
  ! --error 4
未定义的变量:a



-->load('vals.dat','a','b');   // 加载变量 a,b

-->a   // 显示加载结果
 a   =
```

```
    1.    0.
    0.    1.
```

3.4 Scilab 矩阵运算

3.4.1 标量的运算

Scilab 中标量的运算非常容易，主要有加、减、乘、除与乘幂运算。通过“+”、“-”表示加与减；“*”表示乘法；除号分为左除(用“\”表示)与右除(用“/”表示)，在斜杠以下的表示分母；乘幂用“^”表示。Scilab 在变量的运算中要求各元素写在同一行。

```
-->5 + 7
 ans  =

   12.

-->7 - 9
 ans  =

  - 2.

-->9 * 5
 ans  =

   45.

-->9/2
 ans  =

   4.5

-->2\9
 ans  =

   4.5

-->2^5
 ans  =
```

```
32.
```

3.4.2 加法和减法

由于加减运算是将两个矩阵的对应元素逐个相加减，因此要求参与运算的两个矩阵大小相同，用法如下：

C＝A＋B

C＝A－B

特殊情况是其中一个为标量，那么把另一矩阵的每一个元素与该标量相加减。另外“＋”也可作为字符串连接符。如果把和“＋”和“－”作一元运算符使用，则－A表示将矩阵 A 中的每个元素取反。

加法和减法运算：

```
-->A=[5 9 3;0 2 8];B=[1 1 9;6 7 2];

-->C=A+B        //两个矩阵相加
 C  =

    6.    10.    12.
    6.    9.     10.

-->D=C*2        //矩阵与标量相乘
 D  =

    12.   20.    24.
    12.   18.    20.
```

注意：如果输入是字符串矩阵，则“＋”实现字符串的连接，这与数值型矩阵是不同的。如：

```
-->U=['1''2'];V=['3''4'];

-->W=U+V
 W  =

! 13  24  !
```

3.4.3 乘法运算

乘法运算有两种：一种是根据线性代数定义的两个矩阵的相乘，称为矩阵乘，

要求第一个矩阵的列数必须等于第二个矩阵的行数；另一种是两个数组对应元素的相乘，称为数组乘，这种情况下两个矩阵必须有相同的行数与列数。

相应的矩阵乘的计算公式为

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

数组乘的计算公式为

$$C_{ij} = A_{ik} B_{kj}$$

其运算表达式分别记为

矩阵乘：c＝A * B

数组乘：c＝A. * B

矩阵乘和数组乘：

```
-->A=[3 0 5;1 1 2;5 9 6],B=2*ones(3,3)
 A  =

    3.    0.    5.
    1.    1.    2.
    5.    9.    6.
 B  =

    2.    2.    2.
    2.    2.    2.
    2.    2.    2.

-->A*B        //A 和 B 的矩阵乘
  ans  =

    16.   16.   16.
    8.    8.    8.
    40.   40.   40.

-->B*A
  ans  =

    18.   20.   26.
    18.   20.   26.
    18.   20.   26.
```

```
-->A. * B     //A 和 B 的数组乘
 ans  =

    6.    0.    10.
    2.    2.    4.
    10.   18.   12.

-->B. * A
 ans  =

    6.    0.    10.
    2.    2.    4.
    10.   18.   12.
```

从例子可以看出，矩阵乘不满足交换律，而数组乘是满足交换律的。

3.4.4 除法运算

在 Scilab 中定义了两种矩阵除法：矩阵左除与矩阵右除。矩阵除法在线性代数中是没有定义的，只有矩阵逆的定义。矩阵除法的命令为

矩阵左除：c＝A\B

矩阵右除：c＝A/B

如果 A 是一个方阵，那么矩阵左除(A\B)相当于 A 的逆阵左乘 B，即 $A^{-1}B$。这就相当于求方程 Ax＝B 的解。因此如果 A 是一个 n×n 的矩阵，B 是个 n 维列向量或 n×m 的矩阵，矩阵左乘 A\B 返回的就是 Ax＝B 的解。如果 A 是一个m×n 的矩阵，其中 n 近似等于 m，B 是个 m 维列向量或 m×n 的矩阵，那么 x＝A\B 是不定或超定方程的最小二乘解。矩阵右除 A/B 相当于 B 的逆阵左乘 A，即 $B^{-1}A$。

矩阵除法：

```
-->A = [1 2 3;0 1 0;3 2 1];B = [2 3 1]';

-->x = A\B          //矩阵左除
 x  =

 - 1.375
   3.
 - 0.875

-->B = B';
```

```
-->x = B/A          //矩阵右除
 x  =

    0. 125    1. 5    0. 625
```

数组除法也分两种:数组左除与数组右除。数组除法的命令为

数组左除:C=A. \B

数组右除:C=A. /B

数组除法是数组的对应元素相除,因此两个矩阵必须有相同的大小。除非其中一个为标量。其中数组左除是数组 B 和 A 的对应元素相除,即 B_{ij}/A_{ij},数组右除是数组 A 和 B 的对应元素相除,即 A_{ij}/B_{ij} 相除。

数组除法:

```
-->A = [1 5 2;9 7 3];B = [2 2 2;5 5 5];

-->A. \B              //数组左除
 ans  =

    2.          0. 4          1.
    0. 5555556  0. 7142857    1. 6666667

-->A. /B              //数组右除
 ans  =

    0. 5    2. 5    1.
    1. 8    1. 4    0. 6

-->B. /2              //标量运算
 ans  =

    1.      1.      1.
    2. 5    2. 5    2. 5
```

3.4.5 矩阵求逆

矩阵求逆的运算只适合方阵,即行列数相等的矩阵。矩阵求逆的命令为

B=inv(A)

矩阵求逆：

```
-->A=[1 2 3;0 1 0;3 2 1];

-->B=inv(A)
 B  =

 - 0.125   - 0.5    0.375
   0.        1.     0.
   0.375   - 0.5    - 0.125
```

当矩阵是非满秩时，计算将显示出错信息："奇异矩阵"，例如

```
-->inv(ones(5,5))
                ! --error 19
```

问题是奇异的。

3.4.6 乘方运算

C=A^p 矩阵乘方

C=A.^p 向量乘方

乘方运算也分两种：矩阵的乘方运算和数组的乘方运算。

矩阵的乘方运算：c=A^p

数组的乘方运算：c=A.^B

1. 矩阵的乘方运算

矩阵的乘方运算要求A是一个方阵，p是一个标量，如果p是个正整数，A^p是矩阵自乘p次；如果p是个负整数，A^p是矩阵的逆自乘|p|次。如果p不是一个整数，则A^p=V*D.^p/V，其中，V为A的特征向量矩阵；D为A的特征值矩阵；D.^p为数组的乘方。

矩阵的乘方运算：

```
-->A=[1 2 3;3 2 1;1 9 1];

-->A^2                          //相当于A×A
 ans  =

   10.    33.    8.
   10.    19.    12.
   29.    29.    13.

-->A^(-1)                       //结果相当于矩阵求逆
```

```
 ans    =

  - 0. 109375     0. 390625    - 0. 0625
  - 0. 03125    - 0. 03125      0. 125
    0. 390625   - 0. 109375    - 0. 0625

-->A = [1 0 0;0 2 0;0 0 3];

-->A^0. 5
 ans    =

    1.      0.                    0.
    0.      1. 4142135623731      0.
    0.      0.                    1. 7320508075689
```

2. 数组的乘方运算

数组的乘方 C=A. ^B 就是矩阵 A 和 B 的对应元素的乘方，即 $A_{ij}{}^{\wedge}B_{ij}$。A 和 B 必须有相同大小，除非其中一个为标量，这时计算结果仍为矩阵。如果 p 是标量，A 是矩阵，p. ^A 的每一个元素为 $p^{\wedge}A_{ij}$，A. ^p 的每一个元素为 $A_{ij}{}^{\wedge}p$。

数组的乘方运算：

```
-->A = [1 2 3;2 2 2];

-->B = [5 5 5;1 2 3];

-->A. ^B
 ans    =

    1.      32.     243.
    2.      4.      8.

-->2. ^A
 ans    =

    2.      4.      8.
    4.      4.      4.
```

3.4.7 矩阵转置

C = A'

矩阵转置是将一个大小为 n×m 的矩阵转换为 m×n 的矩阵，运算符为单引号“'”。

矩阵转置：

```
-->M=[9 6 4;8 2 3]
 M  =

    9.    6.    4.
    8.    2.    3.

-->N=M'
 N  =

    9.    8.
    6.    2.
    4.    3.
```

3.4.8 矩阵重新定维

1. matrix 函数

用于改变一个向量或数组的维数或大小，但是元素个数保持不变。

函数格式：y=matrix (v,n,m)。

函数功能：将 v 转为 n×m 的矩阵。注意这种转化是按列进行的，即先取原矩阵第一列的数据依次排入新尺寸下矩阵中的第一列数据，如若未排满，依次取原矩阵的下一列数据。所以重新定维前后的矩阵元素个数必须相等，否则出错。

将一个 2×3 的矩阵变换为 3×2 的矩阵。

```
-->A=[1 2 3;9 5 1]
  A  =

    1.    2.    3.
    9.    5.    1.

-->B=matrix(A,3,2)
  B  =
```

```
    1.    5.
    9.    3.
    2.    1.
```

2. size 函数

用于求矩阵或向量的大小。

函数格式：y＝size(x[,sel])。

函数功能：返回矩阵或向量的每一维的大小，结果 y 为一个行向量。如果矩阵 x 是个二维矩阵，则 y 中元素分别代表了该矩阵的行与列的个数。可选项 sel 是个不大于矩阵维数的整数，表示专门对第 sel 维求大小。

求矩阵大小：

```
-->A = [1 2 3;4 5 6]
 A  =

    1.    2.    3.
    4.    5.    6.

-->n = size(A)
 n  =

    2.    3.

-->n = size(A,1)              //求矩阵的行数
 n  =

    2.
```

3.4.9 矩阵提取

1. tril 函数

用于提取下三角矩阵。

函数格式：y＝tril (x[,k])。

函数功能：提取矩阵 x 主对角线以下的元素，其他值为 0。可选项 k 为一个整数，如果 k>0，则在主对角线以上扩大 k 级提取范围；如果 k<0，则在主对角线以下缩小 k 级提取范围。k 的默认值为 0。

2. triu 函数

用于提取上三角矩阵。

函数格式：y＝triu(x[,k])。

函数功能：提取矩阵 x 主对角线以上的元素，其他值为 0。可选项 k 为一个整数，如果 k>0 则在主对角线以上缩小 k 级提取范围；如果 k<0 则在主对角线以下扩大 k 级提取范围。k 的默认值为 0。

提取矩阵的上下三角矩阵：

```
-->A=[1 2 3;4 5 6;7 8 9]
 A  =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.

-->tril(A,-1)
 ans  =

    0.    0.    0.
    4.    0.    0.
    7.    8.    0.

-->triu(A)
 ans  =

    1.    2.    3.
    0.    5.    6.
    0.    0.    9.

-->triu(A,2)
 ans  =

    0.    0.    3.
    0.    0.    0.
    0.    0.    0.
```

3. diag 函数

产生对角矩阵或提取矩阵的对角线元素。

函数格式：[y]＝diag(vm[,k])。

函数功能：

(1) 如果 vm 是一个向量，则产生一个以 vm 的值为对角元素，其他元素的值为 0 的矩阵。参数 k 为可选项，如果 k>0，则 vm 为矩阵主对角线之上的第 k 条对角线；如果 k<0，则 vm 为矩阵主对角线之下的第 k 条对角线。

(2) 如果 vm 是一个矩阵，则提取矩阵的对角元素，产生一个列向量。参数 k 为可选项，如果 k>0，则提取主对角线以上第 k 级的对角线元素；如果 k<0，则提取主对角线以下第 k 级对角线的元素。

产生对角矩阵或提取矩阵的对角线元素：

```
-->u = [9 5 1];

-->diag(u)                    //由向量 u 产生对角阵
 ans  =

    9.    0.    0.
    0.    5.    0.
    0.    0.    1.

-->diag(u,1)                  //由向量产生对角阵，但 u 为主对角线以上的第一级对角线
 ans  =

    0.    9.    0.    0.
    0.    0.    5.    0.
    0.    0.    0.    1.
    0.    0.    0.    0.

-->v = [9 8 7;6 5 4;3 2 1]
 v  =

    9.    8.    7.
    6.    5.    4.
    3.    2.    1.

-->diag(v)                    //从矩阵 v 提取对角线
 ans  =

    9.
    5.
    1.
```

```
-->diag(v, -2)          //提取矩阵 v 的主对角线以下第 2 级对角线
 ans  =

    3.
```

3.4.10 特征值和特征向量

对于一个 n×n 的方阵 A，如果满足以下关系式：

$$Av=\lambda v$$

则 λ 为 A 的特征值，v 为对应于该特征值 λ 的特征向量。Scilab 提供求解矩阵特征值和特征向量的函数，分别为 spec 和 bdiag。

1. 求矩阵特征值

函数格式：evals=spec(A)。

其中

A：实数或复数方阵。

evals：返回的特征值是个实数或复数向量。

2. 求矩阵特征向量

函数格式：[Ab[,x[,bs]]]=bdiag(A)。

其中

A：实数或复数方阵；

Ab：实数或复数方阵为分块对角矩阵，如果每一个子块大小为 1，对角线上的元素为矩阵特征值；

x：实数或复数非奇异矩阵，每一列对应于一个特征向量；

bs：整数向量给出各个子块的大小，缺省各元素值为 1。

该函数其实是用来实现矩阵的分块对角化的。对角化的同时可以给出特征向量。

求矩阵特征值和特征向量：

```
-->A = [6 8 3;9 5 1;7 5 4];

-->ev = spec(A)          //用 spec 函数求 A 的特征值，结果从小到大排列
 ev  =

   16. 060152939141
  - 3. 2235431432294
    2. 1633902040886
```

```
-->[Ab,x,bs] = bdiag(A)      //返回 A 的对角化后的矩阵 Ab,特征向量矩阵 X 以及各子
                               块的大小
 bs  =

    1.
    1.
    1.
 x  =

    0.7006199251844    0.6819472054190    0.0144984387707
    0.6305194824240  - 0.7271834052130  - 0.6587164307835
    0.6680625800576  - 0.1575007429607    1.7380355313516
 Ab  =

    16.060152939141    0.                 0.
    0.                 - 3.2235431432294  0.
    0.                 0.                 2.1633902040886
```

3.4.11 稀疏矩阵

Scilab 可以处理稀疏矩阵，即大多数元素为 0 的矩阵。其算法就是不存储、不操作那些“0”元素，从而达到节省内存与运算时间的目的。因此稀疏矩阵的计算复杂性取决于非零元素的数目，与总的元素个数无关。而常规矩阵是全矩阵，将所有的元素都保存在内存中。

1. 创建稀疏矩阵

1）由全矩阵转化为稀疏矩阵

Scilab 提供将一个全元素存储矩阵转化为稀疏矩阵的函数 sparse。其简略的调用格式为

S=sparse(X)

其中，X 为全矩阵；S 是稀疏矩阵。如果 X 已经是稀疏矩阵，那么函数调用不产生什么影响。但是如果 X 是一个全矩阵，那么函数将把其中的零元素挤掉，只保留那些非零元素。因此当矩阵大而稀疏时，用稀疏矩阵可以有效地节省存储空间。

用 sparse 函数创建稀疏矩阵：

```
-->X=[6 0 9;0 3 0;0 0 7]
 X  =
```

```
    6.    0.    9.
    0.    3.    0.
    0.    0.    7.

-->S = sparse(X)
 S  =

(    3,    3) sparse matrix

(    1,    1)        6.
(    1,    3)        9.
(    2,    2)        3.
(    3,    3)        7.
```

注意，把 s 作为稀疏矩阵首先要报告该矩阵的大小，然后是实际存储的那些非零元素及其所在的行列位置。其逆向的操作，即由稀疏矩阵创建全矩阵的操作是由 full 函数完成的，其调用格式为

X=full(S)

由稀疏矩阵创建全矩阵：

```
-->X1 = full(S)
 X1  =

    6.    0.    9.
    0.    3.    0.
    0.    0.    7.
```

转换后可以直观地检查矩阵存储的非“0”元素和对应位置。

2) 直接创建稀疏矩阵

这种方法仍是调用 sparse 函数，但是参数不同，其调用格式为

S=sparse(index,values)

如果创建有 N 个元素的稀疏矩阵，则 index 为 N×2 的矩阵，index(i,1)与 index(i,2)分别存储了第 i 个元素的行列位置。由 index 存储的最大值决定了该矩阵的大小。values 为 N×1 的向量，是稀疏矩阵元素的值。

直接创建稀疏矩阵：

```
-->row = [1 1 2 3 5 5];          //稀疏矩阵的行号，最大行号为 5

-->col = [2 8 5 6 7 1];          //稀疏矩阵的列号，最大列号为 8

-->index = [row' col']           //合并成下标矩阵
```

```
 index   =

    1.    2.
    1.    8.
    2.    5.
    3.    6.
    5.    7.
    5.    1.

-->val = [3 6 5 2 1 4];                    //6 个元素的值

-->S = sparse(index,val)                   //生成稀疏矩阵
  S   =

(    5,    8) sparse matrix

(    1,    2)        3.
(    1,    8)        6.
(    2,    5)        5.
(    3,    6)        2.
(    5,    1)        4.
(    5,    7)        1.
```

默认情况下产生的矩阵大小由给出的最大行号与列号决定，但是也可以指定矩阵大小。其调用格式为

S=sparse(index,values,dim)

dim 为 1×2 的向量，它给出矩阵的大小。

```
-->A = sparse(index,val,[9,9])
  A   =

(    9,    9) sparse matrix

(    1,    2)        3.
(    1,    8)        6.
(    2,    5)        5.
(    3,    6)        2.
(    5,    1)        4.
(    5,    7)        1.
```

```
-->B = full(A)
 B  =

    0.    3.    0.    0.    0.    0.    0.    6.    0.
    0.    0.    0.    0.    5.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    2.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.
    4.    0.    0.    0.    0.    0.    1.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.
```

用 Size 函数操作可以返回稀疏矩阵的大小。

3）特殊稀疏矩阵

(1) 非零元素都为 1 的矩阵。

通过 spones 函数，可以根据一个已有的稀疏矩阵创建一个结构相同但元素全为 1 的稀疏矩阵，其调用格式为

S1＝spones(S)

创建与已知矩阵位置相同且非零元素都为 1 的矩阵：

```
-->A = [0 9 0 0;6 0 7 0;3 0 0 0;0 8 0 2];

-->S = sparse(A);

-->S1 = spones(S);

-->full(S1)
 ans  =

    0.    1.    0.    0.
    1.    0.    1.    0.
    1.    0.    0.    0.
    0.    1.    0.    1.
```

(2) 稀疏单位阵。

稀疏单位阵即对角元素为 1，其他元素为 0 的单位阵。它可以生成与给定矩阵相同大小的稀疏单位阵，也可以通过输入行列值指定稀疏单位阵的大小。

下面给出利用上面的 S 创建 4×4 的稀疏单位阵的例子：

创建稀疏单位阵：

```
-->S1 = speye(S)
  S1   =

(    4,    4) sparse matrix

(    1,    1)        1.
(    2,    2)        1.
(    3,    3)        1.
(    4,    4)        1.
```

或者直接指定稀疏单位阵的大小：

```
-->S1 = speye(3,4);

-->full(S1)                    //显示全矩阵
  ans   =

    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
```

(3) 稀疏随机阵。

稀疏随机阵为非零元素是 0～1 之间随机数的矩阵，而且随机数的分布由指定的概率值确定。调用格式为

sp＝sprand(nrows,ncols,fill [,type])

sp＝sprand(nrows,ncols,fill)返回一个 nrows 行、ncols 列、大约有 fill * nrows * ncols 个非零元素的稀疏矩阵 sp。其中最后一个可选参数 type 的取值为“uniform”或“normal”，即指定所产生的随机数是服从均匀分布还是正态分布。缺省情况下为均匀分布。

创建稀疏随机阵：

```
-->S = sprand(5,5,0.001);

-->full(S)
  ans   =

   0.6723949727602    0.    0.    0.    0.
   0.                 0.    0.    0.    0.
   0.                 0.    0.    0.    0.
   0.                 0.    0.    0.    0.
```

```
    0.          0.    0.    0.    0.
```

(4) 稀疏全零阵。

稀疏全零阵的所有元素都为零,实际上不占用内存,但足可以用来定义一个稀疏矩阵。创建函数为 spzeros,用法与 speyes 一样。设 S 与上例相同:

创建稀疏全零阵:

```
-->spzeros(S)
  ans   =


(    5,    5) zero sparse matrix
```

相当于

```
-->spzeros(5,5)
  ans   =


(    5,    5) zero sparse matrix
```

2. 稀疏矩阵的查看

1) 稀疏矩阵的查看

对于稀疏矩阵一个基本的要求是查看非零元素的信息。在 Scilab 中用 spget 函数实现。其调用格式为

[index,values,dim]= spget(A)

其中,index 为非零元素的行列信息;values 为非零元素的值;dim 为矩阵的大小。通过调用 spget 函数,这些信息都得到。

查看稀疏矩阵中非零元素信息:

```
-->A=[0 2 0 0;4 0 1 0;0 0 3 0;0 5 0 3];

-->S=sparse(A);

-->[index,values,dim]=spget(S)
  dim   =


    4.    4.
  values   =


    2.
    4.
    1.
```

```
    3.
    5.
    3.
index   =

    1.    2.
    2.    1.
    2.    3.
    3.    3.
    4.    2.
    4.    4.
```

find 函数可以返回非零元素的下标。

```
-->[i,j] = find(S)
  j   =

    1.    2.    2.    3.    3.    4.
  i   =

    2.    1.    4.    2.    3.    4.    2.
```

2) 稀疏矩阵可视化

利用 Scilab 的图形，可以将稀疏矩阵中非零元素的分布以图形方式直观地显示出来。

稀疏矩阵的可视化：

```
-->S = sprand(80,100,0.2,'normal');      //生成约 20 % 的非零元素的随机稀疏矩阵

-->[index,values,dim] = spget(S);

-->x = index( : ,1);y = index( : ,2);

-->plot2d(x,y, - 2)                      //非零元素由“×”符标示
```

3) 稀疏矩阵的计算

稀疏矩阵的计算与常规全矩阵计算方式基本上是一样的。比如矩阵加减法，求矩阵大小等。对于一元函数，如果输入为稀疏矩阵，则输出一般也是稀疏矩阵，如 diag(S)；对于二元运算符，如果两个操作数都是稀疏的，则运算结果也是稀疏的；如果两个操作数一个是稀疏的，另一个是全元素的，除非运算保留稀疏性，否则给出全元素结果。例如，S+A，S * A，S/A 的结果为全矩阵，而 S. * A 的结果为稀

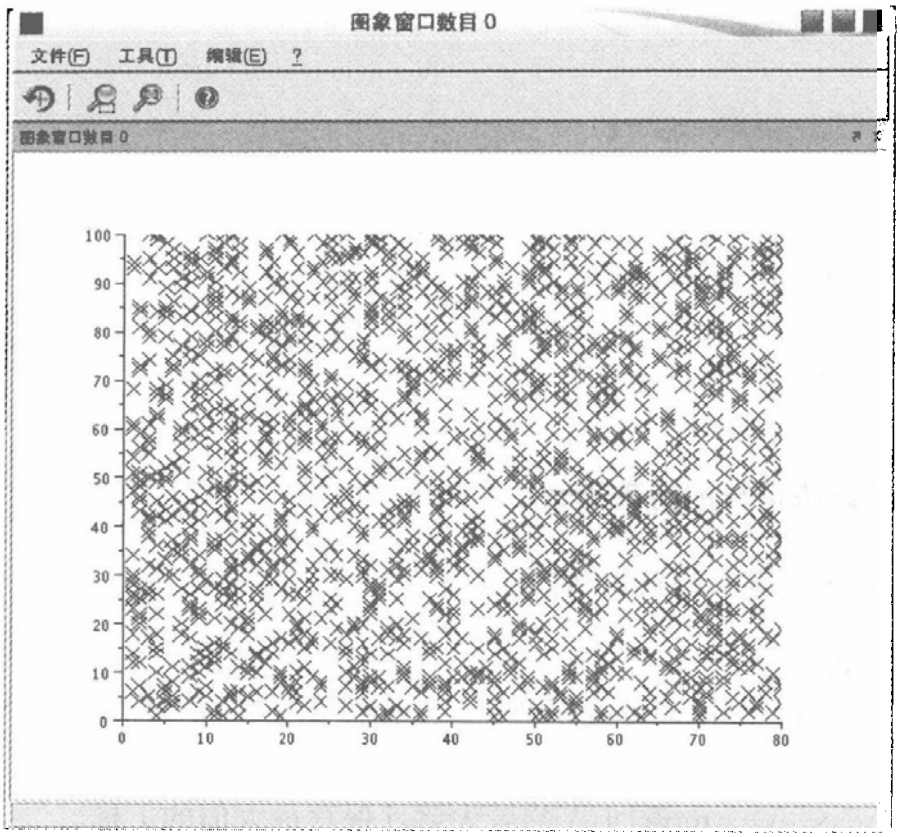


图 3.3 稀疏矩阵可视化

疏的。稀疏矩阵的计算复杂度与矩阵的非零元素个数成正比，但是与矩阵总的元素个数无关。

3. 稀疏矩阵的计算

```
-->S = speye(4,4);                    //定义稀疏单位阵 S

-->A = [1 2 5 3;6 3 3 3;5 2 0 9;2 3 5 3];

-->S(2,4) = 1;                        //定义全矩阵 A

-->S + A
ans  =

    2.    2.    5.    3.
    6.    4.    3.    4.
    5.    2.    1.    9.
    2.    3.    5.    4.
```

```
-->S. * A
ans   =

(    4,    4) sparse matrix

(    1,    1)        1.
(    2,    2)        3.
(    2,    4)        3.
(    3,    3)        0.
(    4,    4)        3.
```

3.4.12 几个矩阵运算的特殊函数

1. 求向量或矩阵中的最大值(max)

调用格式如下：

[m]=max(A) 求向量或矩阵 A 中所有元素的最大值，并将结果返回到 m 中。

[m,i]=max(A) m 返回 A 的最大值，i 保存最大值的下标。

[m,i]:max(A,'r') m 返回一个行向量，其中的元素对应于 A 的每一列的最大值。

[m,i]=max(A,'c') m 返回一个列向量，其中的元素对应于 A 的每一行的最大值。

[m[,i]]=max(A1,A2,…,An) m 的大小与 Aj 相同，各个参数 Aj 为相同大小的矩阵或标量，m 的每一个元素为所有 Aj 对应位置 L 的最大值。i 为取得最大值的 A 的序号 j。

求最大值的例子：

```
-->A=[6 1 1;1 1 2;1 5 1];          //定义矩阵 A

-->m = max(A)                      //返回 A 的最大元素的值
m   =

    6.

-->[m,i] = max(A)                  //返回 A 的最大元素的值及相应位置
i   =

    1.    1.
```

```
m   =

    6.

-->A = [5 4 8; -2 0 9;6 4 -7];

-->[m,i] = max(A,'r')              //返回 A 的每列最大元素的值及相应行号
i   =

    3.    1.    2.
m   =

    6.    4.    9.

-->[m,i] = max(A,'c')              //返回 A 的每行最大元素的值及相应列号
i   =

    3.
    3.
    1.
m   =

    8.
    9.
    6.

-->B = 2 * ones(3,3)
B   =

    2.    2.    2.
    2.    2.    2.
    2.    2.    2.

-->m = max(A,B)                    //求 A 和 B 的最大值矩阵
m   =

    5.    4.    8.
    2.    2.    9.
```

```
    6.    4.    2.
```

求向量或矩阵中的最小值(min)的调用格式与 max 相同。

2. 求向量或矩阵的均值(mean)

调用格式如下：

y=mean(x) 求向量或矩阵 x 中所有元素的平均值，并将结果返回到 y 中。

y=mean(x,r') 返回一个行向量 y，该向量的每一个元素为 x 中对应列的平均值。

y=mean(x,c') 返回一个列向量 y，该向量的每一个元素为 x 中对应行的平均值。

求平均值的例子：

```
-->x=[6 0 5;2 6 9;7 3 2]
 x  =

    6.    0.    5.
    2.    6.    9.
    7.    3.    2.

-->y=mean(x)                    //求 x 所有元素的平均值
 y  =

    4.4444444444444

-->y=mean(x,'r')                //按列求 x 的平均值
 y  =

    5.    3.    5.3333333333333

-->y=mean(x,'c')                //按行求 x 的平均值
 y  =

    3.6666666666667
    5.6666666666667
    4.
```

3. 求元素之和(sum)

调用格式如下：

y=sum(x) 求向量或矩阵x中所有元素的和，并将结果返回到y中。

y=sum(x,dim) 求矩阵x的第dim维中所有元素的和，并将结果返回到y中。

求和的例子：

```
-->x=[6 0 5;2 6 9;7 3 2]
 x  =

    6.    0.    5.
    2.    6.    9.
    7.    3.    2.

-->y=sum(x)                    //求所有元素的和
 y  =

    40.

-->y=sum(x,1)                  //求每一列的和
 y  =

    15.    9.    16.

-->y=sum(x,2)                  //求每一行的和
 y  =

    11.
    17.
    12.
```

4. 求累计和(cumsum)

调用格式如下：

y=cumsum(x) 求向量或矩阵x按列的方向累计的和，并将结果返回到y中。y与x同维。

求累计和的例子：

y=cumsum(x,'r') 在y的每一行保存列方向的累计和。

y=cumsum(x,'c') 在y的每一列保存行方向的累计和。

```
-->x=[1 2 3;4 5 6;7 8 9];
```

```
-->y = cumsum(x)          //按列顺序所有元素累计求和
 y  =

    1.    14.    30.
    5.    19.    36.
    12.   27.    45.

-->y = cumsum(x,'r')      //每列累计求和
 y  =

    1.    2.     3.
    5.    7.     9.
    12.   15.    18.

-->y = cumsum(x,'c')      //每行累计求和
 y  =

    1.    3.     6.
    4.    9.     15.
    7.    15.    24.
```

5. 求矩阵元素的积(prod)

调用格式如下：

y＝prod(x)　求向量或矩阵 x 中所有元素的积，并将结果返回到 y 中。

y＝prod(x,'r')　返回一个行向量 y，其中的元素对应于 A 的每一列的乘积。

y＝prod(x,'c')　返回一个列向量 y，其中的元素对应于 A 的每一行的乘积。

矩阵元素求积的例子：

```
-->x=[9 6 4;1 2 3;6 1 5];

-->y = prod(x)            //所有元素乘积
 y  =

    38880.

-->y = prod(x,1)          //按列求积
```

```
 y  =

    54.    12.    60.

-->y = prod(x,2)      //按行求积
 y  =

    216.
    6.
    30.
```

6. 求累计积(cumprod)

调用格式如下：

y=cumprod(x) 求向量或矩阵 x 按列的方向累计积，并将结果返回到 y 中。y 与 x 同维。

y=cumprod(x, 'c') 在 y 的每一行保存列方向的累计积。

y=cumprod(x, 'r') 在 y 的每一列保存行方向的累计积。

矩阵元素求累计积的例子：

```
-->y = cumprod(x)
 y  =

    9.     324.    2592.
    9.     648.    7776.
    54.    648.    38880.

-->y = cumprod(x,'r')
 y  =

    9.     6.     4.
    9.     12.    12.
    54.    12.    60.

-->y = cumprod(x,'c')
 y  =

    9.     54.    216.
    1.     2.     6.
    6.     6.     30.
```

函数的第二个参数“r”或“c”可以用整数 1 或 2 代替，分别表示行和列。

3.4.13 布尔矩阵

布尔型的数据只有两个值：真或假。

一个关系式的运算结果会返回布尔型数据，它常用于条件判断。布尔型矩阵的操作与普通矩阵相同，如可以相加和转置等。

布尔常量是%t 和%f，输出在屏幕上为“T”和“F”。元素类型全为布尔类型的矩阵为布尔矩阵。顺便介绍以下 Scilab 中的关系运算符：

<	小于	<=	小于等于
>	大于	>=	大于等于
==	等于	<>	不等于

由于关系运算符的优先级别比赋值号“=”要高，所以可以直接给一个关系表达式赋给一个布尔变量，例如，a=c>b，a 为布尔变量。两个相同类型矩阵进行关系运算，生成一个布尔矩阵。也可以通过直接赋值得到。

```
-->%t
 %t  =

  T

-->a=[9 1]>=[3 5]
a   =

  T F

-->[1 3]==1                    //各个矩阵元素与一个标量比较
ans   =

  T F

-->a=1:5
a   =

    1.    2.    3.    4.    5.

-->b=a(a>2)
b   =
```

3. 4. 5.

和其他语言一样，Scilab 有基本的逻辑运算功能。逻辑运算符有

& 逻辑与

| 逻辑或

～ 逻辑非

```
-->A=[%f, %f, %t, %t]
 A  =

  F F T T

-->B=[%t, %f, %f, %t]
 B  =

  T F F T

-->A&B                    //与运算
 ans  =

  F F F T

-->A|B                    //或运算
 ans  =

  T F T T

-->～A                    //非运算
 ans  =

  T T F F
```

3.4.14 多项式矩阵

Scilab 可以方便地处理单变量的多项式计算。这也为完成各种控制问题、信号处理、系统分析等应用提供了方便。下面介绍 Scilab 中几种常用的多项式操作方法。

1. 多项式表示

通常，可以用如下指令定义多项式的变量'x'：

x=poly(0,"x")

Scilab 中的多项式是按升幂顺序显示的。例如，多项式 p 可以表示为

$$1+2x+x^2$$

这种表示方式与一般的计算机表达式不同，但是 Scilab 提供了一个转换函数 poly2str。其调用格式为

[str]=pol2str(p)

该函数将多项式 p 转为字符串 str。

str=pol2str(p)

创建多项式的基本指令如下，其调用格式为

[p]=poly(a,"x",["flag"])

其中

a：实数型矩阵或标量；

x：符号变量；

flag：字符串，值为"roots"或"coeff"，缺省值为"root"。

如果 a 是一个矩阵，那么 p 为特征多项式，即

$$p=\begin{vmatrix} x-a_{11} & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & x-a_{22} & \cdots & -a_{2n} \\ \vdots & \vdots & & \vdots \\ -a_{n1} & -a_{n2} & \cdots & x-a_{nn} \end{vmatrix}$$

如果 v 是个向量，那么 poly(v,"x",["roots"])创建根向量为 v 的多项式。poly(v,"x","coeff")创建系数为 v 的多项式。缺省情况下是按照多项式的根处理。创建一个多项式可以通过多项式的系数，也可以通过多项式的根实现。

2. 通过多项式的根创建多项式

由根向量生成多项式：

以 p=(x−1)(x−2)为例：

```
-->a=[1 2];

-->p=poly(a,"x")
 p  =

                2
    2 - 3x + x
```

或者

```
-->x = poly(0,"x")
 x  =

    x

-->p = 1 + x + 2 * x^2
 p  =

              2
   1 + x + 2x
```

3. 通过多项式系数创建多项式

函数调用格式:[P]=inv_coeff(C[,d,[name]])

其中

C: 多项式系数矩阵;

d: 生成的多项式的最高次,缺省值为 size(c,'c')/size(c,'r')-1,必须是整数;

name: 多项式中变量的名称,如 x,y 等,缺省为 x;

P: 返回的多项式。

通过系数创建多项式

以建立多项式$-3x^2+5x+2$为例:

```
-->C = [2 5 -3];

-->P = inv_coeff(C)
 P  =

              2
   2 + 5x - 3x
```

注意其中 x 的幂的表示形式。相反,也可以通过以下指令得到多项式的系数。

函数调用格式: [C]=coeff(P[,v])

其中

P: 多项式矩阵;

v: 正整数,用来指定返回哪些幂项的系数;

C: 返回的多项式系数。

以 $P=-3x^2+5x+2$ 为例：

```
-->C = coeff(P)          //结果返回一个实数向量
 C   =

    2.    5.   - 3.

-->C = coeff(P,2)        //只返回二次项的系数
C   =

  - 3.
```

4. 求多项式的根

通过调用 Scilab 函数可以求多项式的根，该函数名为 roots，其调用格式为

[x]=roots(p)

其中

p：需要求根的多项式；

x：多项式的根向量。

求多项式的根

以 D=2+3 * x+x^2 为例：

```
-->x = poly(0,"x");

-->D = 2 + 3 * x + x^2;

-->v = roots(p)
 v   =

  - 0.3333333333333
    2.

-->roots(poly(1 : 15,"x"))
 ans   =

    1.
    1.999999999999
    3.0000000000081
```

```
4.000000000231
4.9999999950499
6.0000000441442
6.9999997639129
8.0000008464623
8.9999978742603
10.999995246025
10.000003787072
12.999997690858
12.000004101212
14.000000762698
14.999999888068
```

由此可以看出各个根值的求解精度。

5. 多项式的乘法和除法

当两个多项式相乘时，乘积的系数向量为这两个多项式系数向量的卷积，而除法为系数向量的解卷运算。多项式之间用“*”就可以实现相乘的符号运算，相当于 Matlab 中的 conv 函数。多项式除法用 pdiv 函数实现，相当于 Matlab 中的 deconv 函数。其调用格式为

[R,Q]=pdiv(p1,p2)或[Q]=pdiv(p1,p2)

其中

p1：多项式矩阵；

p2：多项式或多项式矩阵；

R,Q：两个多项式矩阵，分别为商矩阵和余数矩阵。

两个多项式矩阵或者一个多项式矩阵与一个多项式相除，返回商矩阵 Q，或者同时返回余数矩阵 R。矩阵之间满足

$p1_{ij}=Q_{ij}*p2+Q_{ij}$，　当 p2 是一个多项式时

或

$p1_{ij}=Q_{ij}*p2+Q_{ij}$，　当 p2 是一个多项式矩阵时

多项式的乘法与除法运算：

```
-->x = poly(0,"x");          //指定 x 为多项式符号变量

-->x^2 - 3 * x + 2;p2 = 2 * x^3 - x^2 - 1;

-->x = poly(0,"x");

-->p1 = x^2 - 3 * x + 2;p2 = 2 * x^3 - x^2 - 1;
```

```
-->p = p1 * p2                    //多项式 p1 与 p2 相乘,乘积为 p
 p   =

                 2     3     4     5
  - 2 + 3x - 3x + 7x - 7x + 2x

-->[r,q] = pdiv(p2,p1)            //多项式 p2 与 p1 相除,商为 q,余数为 r
 q   =

   5 + 2x
 r   =

  - 11 + 11x
```

比较以'/'进行的相除运算：

```
-->q2 = p2/p1
 q2   =

             2
   1 + x + 2x
   - - - - - - - - - -
   - 2 + x
```

结果为有理式形式,分子分母之间不可再约。从 Scilab 的 denom 与 numer 函数可以获得分式的分子与分母部分：

```
-->denom(q2)          //提取分母
 ans   =

  - 2 + x

-->numer(q2)          //提取分子
 ans   =

             2
   1 + x + 2x
```

多项式矩阵之间的运算遵循矩阵之间加减乘除的运算规则,例如,用"+"实现矩阵对应多项式元素的相加;用".*"实现对应多项式元素的相乘。

多项式矩阵的运算：

```
-->p1 = [x-1 x-2;x+1 x+2];

-->p2 = [x+1 x-1;x x+2];

-->p1 + p2              //多项式矩阵相加
 ans   =

    2x        - 3 + 2x

    1 + 2x      4 + 2x

-->p1. * p2
 ans   =

          2                  2
   - 1 + x      2 - 3x + x

         2                  2
    x + x      4 + 4x + x
```

6. 多项式求导

函数调用格式：pd=derivat(p)。

其中

p：多项式或有理式矩阵；

pd：对 p 的一阶导数。

多项式求导：

```
-->x = poly(0,'x');

-->p = [x^2 + 2 * x + 1   x^3 - 1];     //定义一个最高次为 3 的多项式矩阵

-->derivat(p)
 ans   =

                  2
    2 + 2x      3x

-->p = 1/(x + 1)                        //定义一个有理式
```

```
 p  =

      1
   - - - - -
   1 + x

-->derivat(p)                    //对有理式求导
 ans  =

     - 1
   - - - - - - - - -
             2
   1 + 2x + x
```

7. 多项式求值

Scilab 提供 horner 函数计算给定自变量的多项式的值。如果自变量为一个实数，则计算结果为一个数值，在 Matlab 中功能相似的函数为 polyval；如果自变量是个多项式，则 horner 函数自动进行符号运算，结果仍为一个多项式，在 Matlab 中功能相似的函数为 subs。

调用格式：[v]=horner(p,x)。

其中

p：多项式或多项式矩阵；

x：自变量的取值，可以是实数或多项式；

v：多项式的值，可以为实数或多项式。

多项式求值：

```
-->x=poly(0,'x');          //定义符号变量

-->p=[x-1 x+2];            //建立多项式向量

-->horner(p,2)             //令 x 取值为 2
 ans  =

   1.    4.

-->horner(p,%i)            //令 x 取符号 1
 ans  =
```

```
  - 1. + i       2. + i

-->horner(p,x^2)          //用 x^2 替换 x
 ans   =

          2          2
  - 1 + x     2 + x
```

3.4.15 表类型

前面介绍的各种矩阵内部都是单一类型的数据，但是 list 类型或者说“表类型”的元素可以同时包含各种数据类型，而不是限于单一的某种类型，因此其应用非常灵活。例如，list 类型的元素可以有的为实型数据类型，有的为字符串类型。创建 list 类型数据的指令就是 list，类似于 Matlab 中的 cell 类型。

1. 创建 list 变量

指令格式：list(a1,…,an)。

指令说明：a1,…,an 为加入 list 变量的不同数据，可以为任意类型，包括数组和 list 类型。

建立简单的 list 数据：

```
-->x = list(26,'stringa')          //往 list x 输入一个整数和一个字符串字段
 x   =


       x(1)

    26.

       x(2)

stringa

-->y = list([5 9],x)       //list y 的第一个元素为向量，第二个元素为 list x
 y   =
```

```
      y(1)

   5.    9.

      y(2)


       y(2)(1)

   26.

       y(2)(2)

 stringa
```

这说明 list 类型是可以嵌套使用的。

2. list 数据字段的提取

list 变量的字段可以通过给出字段的索引来调用。

指令格式：[x,y,z,…]=l(v)。

指令说明：提取 list 变量 l 中下标值为向量 v 的数据，将结果赋给相应的变量 x,y,z,…。[x,y,z,…]=l(:)将 list 中的所有元素提取出来放在变量 x,y,z,…中。若要进一步提取 list 字段中的字段，则使用多个“()”进行引用。

list 类型数据的提取，接上例

```
-->a = y(1)
 a   =

    5.    9.

-->b = y(2)(2)
 b   =

 stringa
```

3. list 数据字段的插入

插入字段，其实就是对原来位置字段的重新赋值。

指令格式：l(i)=a。

指令说明:在 list 变量 l 的第一位插入数据 a,原数据被替换掉,list 长度保持不变。若是在表头或表尾插入数据,则 list 长度增加。新的数据的类型可以与原数据类型不同。

list 数据元素的插入:

```
-->l = list(69,'eaefgg');

-->l(1) = 'scilab'              //用"scilab"替换原来的 69
 l  =


      l(1)

scilab

      l(2)

eaefgg

-->[n,s] = l( : )               //提取全部数据放在矩阵中
 s  =

eaefgg
 n  =

scilab

-->l(0) = 'first'               //在表头插入数据
 l  =


      l(1)

first

      l(2)

scilab
```

```
        l(3)

    eaefgg

    -->l(4) = 'last'                    //在表尾增添数据,表长度增 1
     l   =


           l(1)

    first

           l(2)

    scilab

           l(3)

    eaefgg

           l(4)

    last

    -->size(l)
     ans   =

        4.
```

4. list 数据字段的删除

删除字段,用“null”指令。

指令格式：l(i)＝null()。

指令说明：清除 list 变量 l 的第 i 位数据,后面的数据向前移动一位,其他数据保持不变,list 长度减少一位。求 list 元素个数可以用 n＝size(1)或 n＝length(1)。

list 数据元素的删除：

```
-->l = list(9,6,3)
 l  =


       l(1)

    9.

       l(2)

    6.

       l(3)

    3.

-->length(l)        //返回 list 类型元素个数
 ans  =

    3.

-->l(2) = null()    //删除第二个数据
 l  =


       l(1)

    9.

       l(2)

    3.

-->size(l)
 ans  =

    2.
```

5. tlist 和 mlist 类型

tlist 也是 list 类型，但是它的第一个域是个字符向量，定义了该 list 类型定义的对象名称以及各字段的名称，相当于表头。tlist 类型可以替代 Matlab 中的 struct 类型。下面通过例子说明这一类型。

tlist 类型：

```
-->l = tlist(['学生','姓名','成绩'],['王欣','赵明'],[85,92])     //包括姓名、成绩
 l  =


      l(1)

! 学生   姓名   成绩   !

      l(2)

! 王欣   赵明   !

      l(3)

   85.    92.
```

该例中 tlist 的第一个元素为字符串向量，说明了该变量存储的对象是“学生”的信息，信息包括“姓名”和“成绩”；相应地，第二个元素的内容为“姓名”，第三个元素的内容为“成绩”，都是向量。

如果要逐条显示信息，需要通过编写函数实现。关于函数的内容在其他章节有专门叙述，这里只简单给出一个例子：

```
//定义用于改变显示的函数 student
-->l = tlist(['student','name','score'],['王欣','赵明'],[85 92])
 l  =


      l(1)

! student   name   score   !

      l(2)
```

```
! 王欣   赵明   !

       l(3)

    85.     92.

-->function student(l),disp([l.name( : ) string(l.score( : ))]),endfunction

-->student(l)

! 王欣   85   !
!              !
! 赵明   92   !
```

结果为逐条显示每人的姓名和年龄。

可以通过字段名来引用 tlist 中的内容。

```
-->l = tlist(['student','name','score'],['王欣','赵明'],[85 92]);

-->l.name     //通过字段名 name 引用
 ans   =

! 王欣   赵明   !

-->l(2)       //通过下标引用
 ans   =

! 王欣   赵明   !
```

注意，在 Scilab 5.2 中 tlist 的第一个域在定义时，如果字段名采用的是中文字符，则不能够通过字段名引用，但可通过下标引用。

```
-->l = tlist(['学生','姓名','成绩'],['王欣','赵明'],[85,92]);

-->l(3)
 ans   =

    85.     92.

-->l.成绩
  ! --error 7
```

点(dot)不能用来作为该操作符的修饰符。

通过字段名“成绩”引用时会出现如上所示的错误。

mlist类型是tlist类型的一种，但是其对字段的引用不是通过整数下标实现，而是通过字段名实现。下面通过例子说明。

mlist类型：

```
-->l = tlist(['student','name','score'],[' 王欣 ',' 赵明 '],[85 92]);  //定义 tlist
                                                                        类型

-->lm = mlist(['student','name','score'],[' 王欣 ',' 赵明 '],[85 92]);  //定义 mlist 类
                                                                        型

-->l(2)              //引用 tlist 类型的“name”字段
 ans  =

! 王欣   赵明   !

-->lm(2)          //试图通过整数下标引用 mlist 类型的“name”字段，出错
      ! --error 144
给定命令的操作未定义。

请检查或定义函数%l_e来解决重载问题。

-->lm('name')     //通过字段名“name”引用
 ans  =

! 王欣   赵明   !
```

从这个例子可以看出tlist与mlist的区别。

3.5 字 符 串

字符在输入输出信息显示中具有重要作用，几乎是每一种编程语言不可缺少的功能。Scilab提供了字符处理的功能。

在Scilab中，字符串用单引号或双引号将字符括起来表示，例如，'abcdef'或“abcdef”。字符串可以作为矩阵或向量的元素，构成字符串矩阵或向量。下面介绍Scilab中几种常用的操作字符串的方法。

3.5.1 字符串生成

无论是单个字符串还是字符串矩阵或向量都可以通过直接赋值产生。矩阵的

赋值方法与前面介绍的相同。

```
-->s = "abcdef"          //单个字符串直接赋值
 s  =

 abcdef

-->sm = ['this','is';'string','matrix']        //产生字符串矩阵
 sm  =

! this    is      !
!                 !
! string  matrix  !
```

获得字符串的间接方法是通过函数间接转换得到，选择哪一种方法可根据具体使用场合而定。可以实现这一目的的函数有以下几种。

code2str：将整型 Scilab 代码转换为相应的字符，问题是这种转换后的编码不是一般的 ASCII 编码。与 code2str 功能相反的函数为 str2code。

string：将矩阵转换为字符类型。

emptystr：产生空字符串。

```
-->s = code2str([9 5 7 2 4])              //通过代码得到相应字符
 s  =

95724

-->s = string(rand(1,2))                  //将实数向量转为字符串向量
 s  =

! 0.0042472956702   0.7690750225447   !
```

在最后一个例子中虽然结果看起来还是一个实型向量，但实际上主字符型向量，在显示时与实型或整型数据不同的是字符型数据按实际宽度显示。比较下面两种变量的默认显示：

```
-->a = '3.1415'           //字符串
 a  =

3.1415

-->a = 3.1415             //实型数据
```

```
 a   =

    3.1415

-->s = emptystr(2,3)          //产生空字符串
 s   =

!        !
!        !
!        !
```

3.5.2 字符串连接

通过字符串连接可以将短的字符串组合成长的字符串。如前所述，用加号＋可以实现这一功能。当参与运算的是两个大小相同的字符串数组时，对应位置的字符串元素相连接，产生一个新的字符串数组。与字符串连接有关的运算符与函数如下。

＋：将两个字符串左右相连。如果参与运算的是两个大小相同的字符串矩阵，则将对应位置上的字符串连接生成一个新的字符串数组。

函数 strcat：调用格式为

txt＝strcat(vstr[,strp])

函数功能为将一个字符串数组 vstr 中的各个元素连接起来形成单个字符串 txt。在 strcat 函数中增加第二个可选字符串类型参数 strp 可以在各个元素间插入一个字符串，以起到隔离数据的作用。

```
-->x = 'scilab';y = 'programming';z = x + y          //用“ + ”连接两个字符串
z   =

scilabprogramming

-->s1 = string(zeros(2,2));s2 = string(ones(2,2));

-->s = s1 + s2              //“ + ”连接两个字符串数组
s   =

! 01  01  !
!         !
! 01  01  !
```

```
-->s = strcat(['scilab' 'linux'])        //字符串数组内元素的连接
s  =

scilablinux

-->strcat(string(1:8),',')              //在数组元素之间插入“,”号
ans  =

1,2,3,4,5,6,7,8
```

函数调用格式：n＝length(M)。

该函数可以返回每个字符串的长度。如果参数为字符串矩阵，则结果不再是字符串的长度，而是该字符串矩阵的大小。这与对实数矩阵的操作类似。可以实现此功能的函数还有 size 函数。

```
-->length('scilab')
ans  =

    6.

-->size(['scilab' 'linux'])     //返回字符串矩阵的大小
ans  =

    1.    2.

-->length(['scilab' 'linux'])     //返回字符串矩阵元素的长度
ans  =

    6.    5.
```

3.5.3 大小写转换

函数调用格式：[y]＝convstr(str_matrix,[“flag”])。

该函数可以将字符串全部转换为大写或小写。flag 为转换标志，当 flag＝‘u’时，所有字符转换为大写；当 flag＝‘l’时，所有字符转换为小写。

字符串大小写转换

```
-->A = ['Scilab' 'on' 'Linux']
```

```
A  =

! Scilab  on  Linux  !

-->convstr(A,'u')
ans  =

! SCILAB  ON  LINUX  !

-->convstr(A,'l')
ans  =

! scilab  on  linux  !
```

3.5.4 字符串的查找

函数调用格式：ind=strindex(strl,str2)。

该函数可以查找一个字符串或字符串数组 str2 在另一字符串中的位置 strl，并将 str2 每次在 strl 中出现的首字母位置保存在向量 ind 中。如果 str2 为字符串数组，则分别寻找各元素的出现位置。若不存在则返同空值。

```
-->k = strindex('SCI/modules/intersci','/')        //"/"在字符串中出现两次，查询其
                                                     位置
k  =

    4.    12.

-->strindex('中国','国')
ans  =

    2.

-->k = strindex('SCI/modules/intersci',['SCI','sci'])    //分别返回 SCI 与 sci 的起
                                                           始位置
k  =

    1.    18.

-->strindex('SCI/modules/intersci','scilab')     //'scilab'在原字符串中不存在，给
```

出空值

```
ans  =

    []
```

3.5.5 字符串替换

函数调用格式：str＝strsubst(strl,str2,str3)。

该函数将原字符串 strl 中的一部分 str2 用另一个字符串 str3 替换掉。如果 strl 不包含 str2，则 strl 保持不变；否则 str2 部分被 str3 替代。

```
-->strsubst('Scilab 科学计算自由软件 ','科学 ','矩阵 ')  //将其中的“科学”用“矩
                                                          阵”代替
ans  =

Scilab 矩阵计算自由软件
```

利用字符串实现公式的赋值计算：

函数调用格式：A＝evstr(str)。

该函数将原字符串 str 中的各个已知变量的数值代入公式进行计算。

```
-->A = ['2 * x - y' 'p^q' '5 * t^3 + 2' 'sin(w)']
A  =

! 2 * x - y   p^q   5 * t^3 + 2   sin(w)   !

-->x = 1;y = 6;p = 2;q = 3;t = 3;w = %pi;

-->evstr(A)
 ans  =

  - 4.    8.    137.    1.224646799D - 16
```

Scilab 程序设计

4.1 Sce 文件

到目前为止一直在 Scilab 主窗口中介绍命令。现在将介绍如何使用 Scilab 文件中的命令。文件位于硬盘上的某个地方，需要用一些命令，找到并使用它们。要创建和编辑文件，用户可以使用 Scilab 自带的 Scilab text editor 编辑器或其他的任何文本编辑器。Scilab 能够识别到当前文件的读写目录，通过 Scilab 的一些命令就可以来浏览目录：

```
-->SCI                          // Scilab所在的目录
 SCI  =

/opt/scilab-5.2.2/share/scilab

-->pwd                          //当前读写文件的目录
 ans  =

/home/scilab/文档

-->chdir('..')                  //移动到上一级目录
 ans  =

  T

-->pwd                          //移动后的目录
 ans  =

/home/scilab

-->chdir('文档')                //返回原来的目录
 ans  =
```

```
 T
```

还可以在 Scilab 的主窗口中执行 Shell 命令来对目录进行操作。

```
-->unix_w('mkdir test')          //创建目录 test

-->chdir('test')                 //进入目录 test
 ans  =

  T

-->unix_w('ls -l')               //显示 test 目录的内容
总计 0
```

由此可见,创建一个目录并进入它是非常简单的。现在在 Scilab text editor 中用下面的代码编辑和生成“test. sce”。(将其保存在刚刚创建的目录“/home/scilab/test”中)。

```
message = '学习 Scilab';
```

在 Scilab text editor 中输入完成这段代码后,点击保存按钮,将其命名为“test. sce”。然后就可以在 Scilab 主窗口中执行该文件了。

```
-->exec('/home/scilab/test/test. sce');

-->message
message  =

学习 Scilab
```

也可以通过执行主窗口中的“文件”→“执行”命令,打开“执行文件”对话框,在其中选择 test. sce 文件,点击“确定”来执行该文件。

这种类型的编程有一些缺点。正如所看到的,所有在文件中定义的变量都是“事后维护”,即没有经过预先的声明。我们试图使代码尽可能简单,让少数几个变量同时在内存中进行运算。这将比当文件被执行,而文件中的变量没有被激活更好。其次,如果将各种编程的方法放在一起(即编写到同一个文件),那么文件中的对象就会不明确。最后,因为没有定义该文件的输入,如果不读代码,就不知道哪

些变量将会对结果产生影响。对于编写一些小程序，这些不是很重要，但如果要做更大的项目这就显得至关重要。

通常情况下，Sce 文件是 Scilab 程序的主要部分，但我们需要一种封装代码的方式，通过这种方式能够更好地定义程序的输入和输出，并且变量在执行后不用保留。换句话说，需要利用定义“函数”的方法来编写各种各样类型的程序。

4.2 程序控制结构

4.2.1 选择结构

很多情况下，经常会遇到根据某个条件的成立与否而做出相应的选择，或是从多个可能中选择一个，这就需要依据不同的条件执行不同的语句，在编程语言里，是通过选择结构实现的。Scilab 的选择结构语句主要有 if 语句和 select-case 语句。

1. if 语句

当关键词 if 与指令分行写时，指令的基本格式如下：

```
if  条件表达式
    语句序列;
end
```

该语句的功能：如果条件表达式结果为真，则执行后面的语句序列，如果表达式结果为假时，就直接执行 end 以后的语句(图 4.1)。

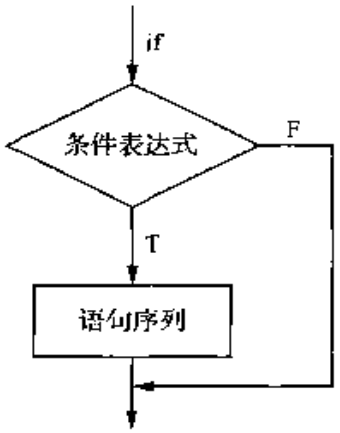


图 4.1 不完整 if 语句流程图

例如，下面的例子可以用来判断输入的数是否是非负的。

```
a = input("a : ");
if a<0
```

```
  disp("a 必须是大于或等于 0 的数!");
end
```

如果条件表达式结果为假时,还需要执行指定的语句,可以采用如下的格式:

```
if 条件表达式
  语句序列 A;
else
  语句序列 B;
end
```

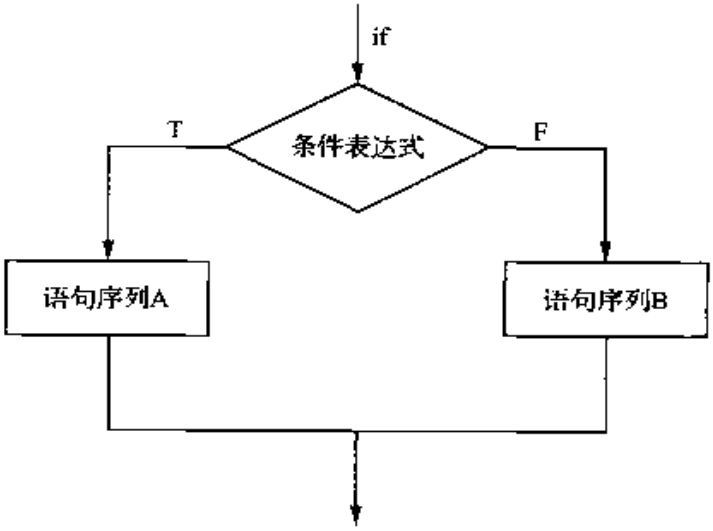


图 4.2 完整的 if 语句流程图

当表达式结果为假时执行 else 后面的语句(图 4.2)。这种书写格式与 Matlab 一样。但是当指令与 if 语句写在同一行时,表达式后面必须有 then 或逗号",",例如:

```
if 表达式 then  语句序列;end
if 表达式,     语句序列;end
```

判断语句与循环语句结合在一起使用可以灵活地解决问题,以 if 语句可以实现 for 循环和 while 循环的合理跳出或中断。

以 for 循环求容差变量 EPS。

```
EPS = 1;
for n = 1 : 100
   EPS = EPS/2;
   if (1 + EPS)< = 1
        EPS = EPS * 2
        break
    end
```

```
end
num = n - 1
```

运行结果如下：

```
 EPS   =
    2.220D - 16


num   =
    52.
```

本例 for 循环的循环次数要足够大(大于 53)；if 语句检验 EPS 是否变得足够小，以至于可以看成是 0，如果是则使 EPS 乘 2，break 命令强迫跳出 for 循环，转到循环外的下一个语句。如果一个 break 语句出现在一个嵌套的 for 循环或 while 循环结构里，那么只跳出 break 所在的那个循环，不跳出整个嵌套循环。

if 语句的嵌套形式为

```
if 表达式 1 then 语句序列 A
  else 表达式 2 then 语句序列 B
  .
  .
  .
  else 语句 N
end
```

注意每个 if 必须有一个 end 与之配对，否则出错。

使用 for 语句和 if 语句创建以下矩阵：

$$A=\begin{bmatrix}9 & 2 & 0 & 0 & 0\\ 2 & 9 & 2 & 0 & 0\\ 0 & 2 & 9 & 2 & 0\\ 0 & 0 & 2 & 9 & 2\\ 0 & 0 & 0 & 2 & 9\end{bmatrix}$$

```
A = [ ];
for k = 1 : 5
  for j = 1 : 5
    if k = = j
      A(k,k) = 9;
    elseif abs(k - j) = = 1
      A(k,j) = 2;
    else
      A(k,j) = 0;
```

```
    end
  end
end
A
```

运行结果如下：

```
A  =

    9.    2.    0.    0.    0.
    2.    9.    2.    0.    0.
    0.    2.    9.    2.    0.
    0.    0.    2.    9.    2.
    0.    0.    0.    2.    9.
```

2. select_case 语句

select case 语句用于有多种备选的情况，类似于 Matlab 中的 switch case 语句。与前面 if 语句相同，当指令与 case 语句写在同一行时，必须加上 then 或“,”，其语句格式如下：

```
select 表达式 0,
case 表达式 1 then 语句 A,
case 表达式 2 then 语句 B,
        .
        .       .
        .
case 表达式 n then 语句 N,
[else 其他语句],
end
```

语句说明：当表达式的值等于表达式 1 的值时，执行语句 A；当表达式的值等于表达式 2 的值时，执行语句 B；…；当表达式的值等于表达式 n 的值时，执行语句 N；当表达式的值不等于任何 case 后面所列的表达式时，执行其他语句。当任何一个分支语句执行完后，都直接转到 end 语句的下一条语句。

在 Matlab 中，switch 表达式的类型只能是标量或字符串，但在 Scilab 中，select表达式可以是任意类型，关键词 otherwise 用关键词 else 替代。

使用 select-case 语句完成考生成绩 score 的转换：

① score ≥90 分，优；② 90＞score ≥80 分，良；③ 80＞score≥70 分，中；④ 70＞score ≥60 分，及格；⑤ 60＜score，不及格。

```
score = input('请输入考生成绩:score = ');
```

```
select fix(score/10)
  case 9
    grade = '优'
  case 8
    grade = '良'
  case 7
    grade = '中'
  case 6
    grade = '及格'
  else
    grade = '不及格'
end
```

运行结果如下：

```
请输入考生成绩:score = 79
grade   =

 中
```

4.2.2 循环结构

循环结构是结构化程序设计的一种基本结构，它通常用于处理大量的有规律的重复操作或运算。与 Matlab 一样，Scilab 可以通过 for 语句和 while 语句实现循环，而且格式也很相似。不同的是，当 for 语句与循环体写在同一行时，在循环条件后面必须有关键词 do 或者逗号“,”。

1. for 语句

语句格式：

```
for 循环变量=初值：步长：终值
   语句序列(循环体)
end
```

循环变量步长的缺省值为 1(即当省略步长时，步长的值为 1)，也可以自己指定或正或负的实数。每执行循环体一次，循环控制变量的值将增加步长大小，当循环控制变量的值大于终值时循环结束。每一个 for 与一个 end 对应。在 for 循环中，循环体内不能出现对循环控制变量的重新设置，否则将会出错；for 循环允许嵌套使用。下面是使用 for 语句的几个例子。

利用 for 语句求 1+2+3+…+100 的和。

```
-->s = 0;
```

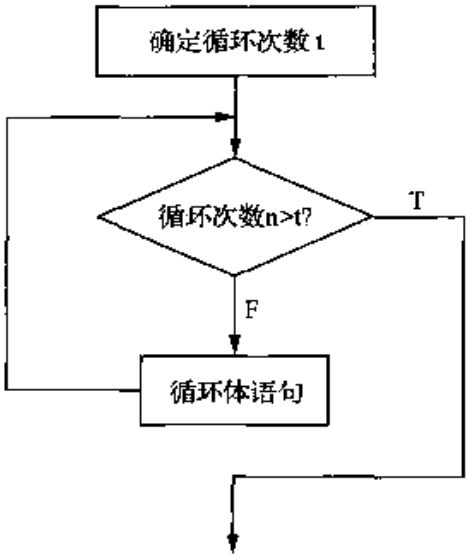


图 4.3 for 语句流程图

```
-->for i = 1 : 100
-->s = s + i;
-->end

-->s
 s    =

    5050.
```

也可以将 for 语句写在一行，用“,”隔开，如下例：

```
-->s = 0;

-->for i = 1 : 100, s = s + i; end

-->s
 s    =

    5050.
```

还可以将步长设置成负数，此时步长是递减的，如下例：

```
-->s = 0;

-->for i = 100 : -1 : 1
-->s = s + i;
-->end
```

```
-->s
 s   =

    5050.
```

for 语句还可以嵌套使用，如下例：

利用 for 语句求 1！＋2！＋3！＋…＋10！。

```
-->m = 0;

-->for i = 1 : 10
-->n = 1;
-->for j = 1 : i
-->n = n * j;
-->end
--> m = m + n;
-->end

-->m
 m   =

    4037913.
```

比较下面两段程序的执行情况。

```
(a) -->for n = 1 : 10
-->x(n) = sin(n * %pi/10);
-->end

-->x
 x   =

    0. 3090169943749
    0. 5877852522925
    0. 8090169943749
    0. 9510565162952
    1.
    0. 9510565162952
    0. 8090169943749
    0. 5877852522925
    0. 3090169943749
```

```
    1.224646799D-16
(b) -->n=1:10;

-->x=sin(n* %pi/10);

-->x
 x  =


           column 1 to 5

   0.3090169943749    0.5877852522925    0.8090169943749    0.9510565162952
1.

           column 6 to 9

   0.9510565162952    0.8090169943749    0.5877852522925
0.3090169943749

           column 10

    1.224646799D-16
```

比较两段程序的运行结果，可以看出结果相同，但后者执行更快，更直观、简便。因此，当有一个等效的数组方法来解给定的问题时，应避免用 for 循环。

为了得到更快的速度，在循环被执行之前，需要对数组进行预先分配。如(a)，在 for 循环内每执行一次命令，变量 x 的大小增加 1，迫使 Scilab 每进行一次循环都要花费时间对 x 分配更多的内存。为了省去这个步骤，可以在(a)程序的首行加入：

```
-->x=zeros(1,10);      //为 x 分配内存单元
```

此外，矩阵的列数与 list 变量的元素个数也可做循环次数控制。

用 for 循环求行向量[-1,9,8,3,-5]各元素之和。

```
-->x=[-1,9,8,3,-5];

-->s=0;

-->t=0;

-->for i=x
```

```
-->i                    //显示每一次循环变量的值
-->t = t + 1;           //记录循环次数
-->s = s + i;           //计算行向量 a 各元素之和
-->end
 i  =


  - 1.
 i  =

    9.
 i  =

    8.
 i  =

    3.
 i  =

  - 5.

-->t,s                  //显示总的循环次数和计算结果
 t  =

    5.
 s  =

    14.
```

可以看出，总循环次数为 5，第 i 次循环时循环变量的值为 x(i)，计算结果为行向量 x 各元素之和。

2. while 语句

语句格式：

while 表达式

循环体

end

首先判断表达式，如果表达式值为真，则执行循环体部分一次；再来判断表达式是否为真。这样重复进行，一直到表达式值为假，就跳过循环体部分，向下继续

执行。如果表达式开始时为真，为避免死循环，循环体内应有改变表达式值的语句。While 循环可以用于循环次数事先未知的情况。

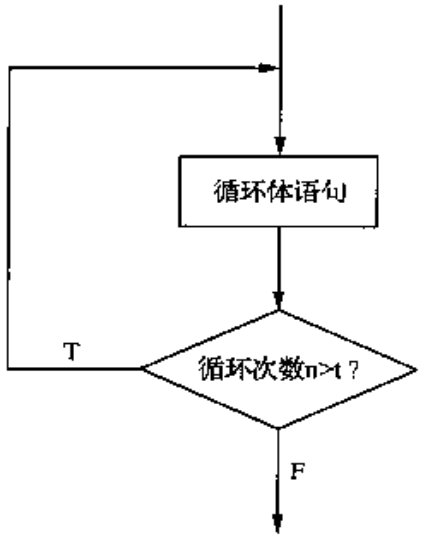


图 4.4 while 语句流程图

求平方值小于 1000 的最大整数

```
-->i = 1;

-->while i * i<1000
-->i = i + 1;
-->end

-->i
 i   =

    32.
```

利用 while 循环可以测算计算机的浮点运算精度。如下：

```
-->n = 0;eps = 1;

-->while 1 + eps>1
-->eps = eps/2;
-->n = n + 1;
-->end

-->n = n - 1,eps = eps * 2
 n   =
```

```
    52.
eps   =

    2.220446049D-16
```

eps 从 1 开始，只要当 1+eps>1 为真，就一直执行 while 循环体内的语句。由于 eps 不断地被 2 除，eps 逐渐变小，当它小于 2.220446049D-16 被看成 0，从而使 eps+1 不大于 1，于是 while 循环结束。因为循环条件是 1+eps>1 为假时才跳出循环，所以 eps 应取使得 1+eps>1 为假的前一次结果（实质上是求满足 1+eps>1 的最小的 eps，即计算机能够区分 1+eps 与 1 为不同值的最小 eps），因此最后的结果 n 要减 1，eps 要与 2 相乘。从运行的结果可以看出，在进行了 52 次循环后得到的 eps 为 2.220446049D-16。

4.3 函数文件

函数文件的扩展名也是“.sci”，其最重要的一条是文件的第一句可执行语句为 function 引导的定义行。紧跟的通常是“//”引导的注释语句，说明函数的功能、作者等相关信息。然后是函数体。以下为 Scilab 提供的提取稀疏矩阵、有理数矩阵等的对角矩阵函数，文件名为“%sp_diag.sci”。此文件在/opt/scilab-5.2.2/share/scilab/modules/overloading/macros/目录下可以找到。

```
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) INRIA
//
// This file must be used under the terms of the CeCILL.
// This source file is licensed as described in the file COPYING, which
// you should have received as part of this distribution.  The terms
// are also available at
// http://www.cecill.info/licences/Licence_CeCILL_V2-en.txt

function d= %sp_diag(a,k)
// %sp_diag - implement diag function for sparse matrix, rational matrix ,..

[lhs,rhs] = argn(0)
if rhs == 1 then k = 0,end

  [ij,v,sz] = spget(a)
  m = sz(1);n = sz(2)
  if m>1&n>1 then
```

```
    l = find(ij(:,1) = = (ij(:,2) - k))
    if k< = 0 then
      mn = mini(m + k,n)
      i0 = - k
    else
      mn = min(m,n - k)
      i0 = 0
    end
    kk = abs(k)
    if l = = [] then d = sparse([],[],[mn,1]);return;end
    d = sparse([ij(l,1) - i0,ones(ij(l,1))],v(l),[mn,1])
  else
    if m>1 then ij = ij(:,1);else ij = ij(:,2);end
    nn = max(m,n) + abs(k)
    if ij = = [] then
      d = sparse([],[],[nn,nn])
    else
      if k>0 then
    d = sparse([ij,ij + k],v,[nn,nn])
      else
    d = sparse([ij - k,ij],v,[nn,nn])
      end
    end
  end
endfunction
```

上例中的函数定义行为

```
function d = % sp_diag(a,k)
```

当函数有多个输入或输出参数时，参数之间用逗号隔开，多个输出参数用方括号[]括起来，如 function [min,txt,top]=sp_min()。如果没有输入或输出，可以不写相应的参数，如 function=%i_plot2d(varargin)。

Scilab 本身提供了很多工具箱，这些几乎都给出这样的函数文件，位于“macro”目录下。与 Matlab 不同的是，Scilab 定义的函数不能直接使用，在第一次调用前必须先用“exec”命令将其加载到内存中，以后可以重复调用。函数的定义与使用如下。

(1) 定义一个简单的函数，输入为一个正数(圆的半径)，返回该圆的面积。

```
function s = test_area(r)
//Scilab 函数测试,用于求圆的面积
```

```
a = %pi * r * r;
endfunction
```

将这些内容保存在文件“test_fun. sci”中。文件保存的路径可以在 Scilab text editor 的标题栏中显示，如图 4. 5 所示。

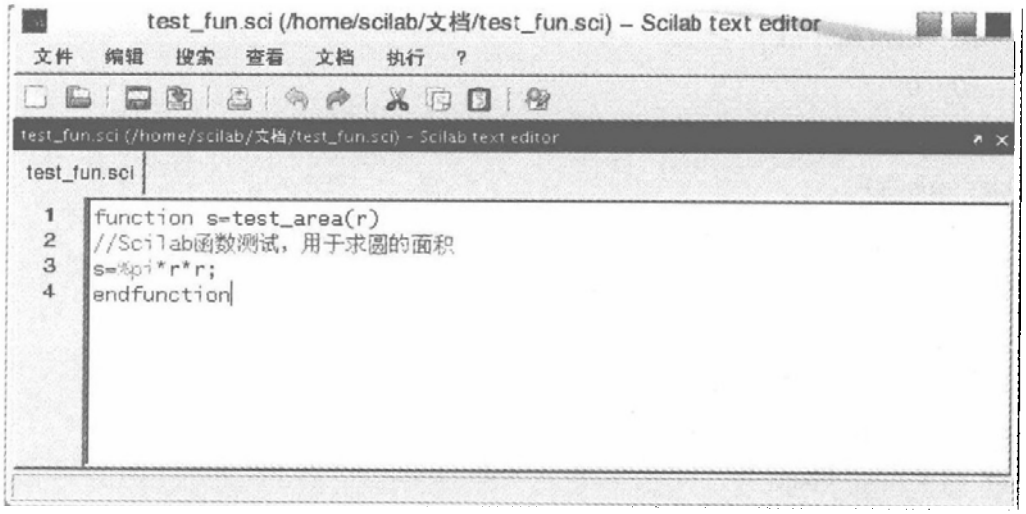


图 4. 5 Scilab 文本编辑界面

(2) 加载以及调用函数。

```
-->exec('/home/scilab/文档/test_fun. sci');      //加载该函数于内存中
-->a = test_area(3)          //调用该函数，得到返回结果
 a   =


    28. 274334
```

注意这里加载函数使用的是 exec 命令，这与 Scilab 4 下使用的 getf 命令有所区别。在 Scilab 5 中如果使用 getf 命令，系统会给出如下的警告：

```
-->getf('/home/scilab/文档/test_fun. sci');
警告：函数 getf 已废弃。
警告：请用 exec 代替。
警告：这个函数将从 Scilab 5. 3 中永久地移除。
```

因此在用 Scilab 编程时通常编写一个“loader. sci”，用来加载所有要用到的自定义函数。

另外一种定义函数的方法是用关键字在线定义，在这种情况下不需要用“exec”加载，可直接使用。使用 functlon/endfunction 可以在程序的任意位置定义函数，使用起来非常方便。

```
-->function y = sq(x),y = x^2,endfunction


-->a = sq(2) + 1
```

```
a   =

    5.
```

要注意，如果在调用的时候不加()，则结果只是复制一个函数。

```
-->s = sq                    //复制函数 sq,生成功能相同的函数 s
s   =

[y] = s(x)

-->b = s(3) + 2              //调用新函数 x
b   =

    11.
```

Scilab 支持函数的递归调用，但是递归的深度有限制。

还有一种定义函数的方法是使用 deff 方式进行在线定义，利用这种方法定义的函数通常是定义在一个函数文件中，一般形式为：

deff('[s1,s2,...]=newfunction(e1,e2,....)',text [,opt])

其中，newfunction 表示被定义函数的函数名，由用户自己设定，可以是不同于 Scilab中的关键字与已定义的函数名的以英文字母开头的英文字母与数字组合而成的字符串，即标识符。e1,e2,.... 表示函数的输入变量；s1,s2,... 表示函数的输出变量。text 为字符串矩阵，通常是字符串向量，它的每一个元素是命令或语句，即 text 的内容是以字符串形式表示的命令或语句。opt 是可选字符串，它的值可以选择'c'、'p'和'n'。其中'c'表示该函数将被更有效地编译；'p'表示函数将被编译并且准备分析(函数的分析 profile 主要功能是当一个函数被执行时，系统会计算每一行执行有多少时间以及每一行执行可能花费的 CPU 时间)；'n'表示函数将不会被编译。opt 选项默认值为'c'。

尽管这两种类型的文件("sce"文件和"sci"文件)都包含代码，但它们是有很大差别的。"sce"文件可以直接执行，而"sci"文件需要先加载然后再执行。两种类型文件的不同不只是文件扩展名的差别，其内部结构和使用时访问它们的命令也是不同的。当定义函数，输入一些检查有助于代码的安全性。在函数内部，使用命令 argn(0)可以返回该函数的输入和输出参数的个数。Scilab 中支持可选参数的功能(在 Scilab 的帮助文档中可选参数用括号加以标识)，当调用函数的时候，如果参数没有赋值，Scilab 取默认(default)值。检查 Scilab 函数的步骤大体上包括：

(1) 利用 argn(0)函数检查输入变量的个数；

(2) 利用 type 函数检查输入变量的类型；

(3) 利用 size 函数检查的输入变量(标量、向量或矩阵)的结构。

4.4 局部变量与全局变量

无论在脚本文件还是在函数文件中,都会定义一些变量。函数文件所定义的变量是局部变量,这些变量独立于其他函数的局部变量和工作空间的变量,即只能在该函数的工作空间引用,而不能在其他函数工作空间和命令工作空间引用。但是如果某些变量被定义成全局变量,就可以在整个 Scilab 工作空间进行存取和修改,以实现共享,即全程有效。因此,定义全局变量是函数间传递信息的一种手段。

在函数文件中定义全局变量时,如果在当前工作空间已经存在相同的变量,系统将会给出警告,说明由于将该变量定义为全局变量,可能会使变量的值发生改变。为避免发生这种情况,应该在使用变量前先将其定义为全局变量。

在 Scilab 中,全局变量用命令 global 定义。函数文件内部的变量事实局部变量,它们与其他函数文件及 Scilab 工作空间相互隔离。但是,如果在若干函数中都把某一变量定义为全局变量,那么这些函数将公用这一个变量。全局变量的作用域是整个 Scilab 工作空间,即全程有效。所有的函数都可以对它进行存取和修改,以实现共享。因此,定义全局变量是函数见传递信息的一种手段。

用命令 global 定义全局变量,其格式为

global A B C

将 A、B、C 这 3 个变量定义为全局变量。

需要指出,在程序设计中,全局变量固然可以带来某些方便,但却破坏了函数对变量的封装,降低了程序的可读性。因而,在结构化程序设计中,全局变量是不受欢迎的。尤其当程序较大,子程序较多时,全局变量将给程序调试和维护带来不便,故不提倡使用全局变量。如果一定要用全局变量,最好给它起一个能反应变量具本含义的名字,并且一般用大写字母表示,以免和其他变量混淆。

全局变量应用的例子。

先建立函数文件 wadd.sci,该函数将输入的参数加权相加:

```
function  f  = wadd(x,y)
                //给两个变量进行不同的加权后相加
                global  ALPHA BETA
                f  =  ALPHA * x + BETA * y;
endfunction
```

在 Scilab 主窗口中输入:

```
-->global ALPHA BETA
```

```
-->ALPHA = 1;

-->BETA = 2;

-->s = wadd(3,2)
```

输出为：

```
 s  =

    7.
```

由于在函数 wadd 和基本工作空间中都把 ALPHA 和 BETA 两个变量定义为全局变量，所以只要在命令窗口中改变 ALPHA 和 BETA 的值，就可改变加权值，而无须修改 wadd. sci 文件。

上例只在函数 wadd 和命令窗口中把 ALPHA 和 BETA 变量定义为全局变量，在实际编程时，可在所有需要调用全局变量的函数里定义全局变量，这样就可实现数据共享。为了在基本工作空间中使用全局变量，也要定义全局变量。

在函数文件里，全局变量的定义语句应放在变量使用以前，为了便于了解所有的全局变量，一般把全局变量的定义语句放在文件的前部。

4.5 程序举例

下面给出一个用于计算一个给定向量平均值的函数例子。

```
function [y] = average(x)
  //求向量元素的平均值
  //输入 x：一个向量，如果输入不是向量，则会导致错误提示
  //输出 y：向量 x 的元素的平均值
  [m,n] = size(x);
  if (~((m = = 1)|(n = = 1))|(m = = 1)&(n = = 1))    //判断输入 x 是否为一个向量
    error('输入必须为一个向量！')
  end
  y = sum(x)/length(x);    //实际计算，求平均值
endfunction
```

当指定输入参数值后，在窗口中运行：

```
-->z = 1 : 99;

-->average(z)
 ans  =
```

```
    50.
```

当输入不是向量，会提示错误，如下：

```
-->x=[1,3;5,7];

-->average(x)
! --error 10000
输入必须为一个向量!
at line       7 of function average called by :
average(x)
```

4.6 程序调试

Scilab 中调试程序时，可以使用 disp，pause，resume，return 与 abort 等命令来帮助用户进行调试。

1. disp

在调试程序的过程中，为显示中间结果，可以用 disp 指令，其指令格式为

```
disp(x1,[x2,…,xn])
```

其中，xi 为任意类型的数据，一次可以显示多个数据。

2. pause

在代码的某个位置加上 pause 指令后可以使程序运行暂时中断，并进入新的高一层的窗口界面，这时 Scilab 的提示符变成另外一种状态，显示当前的级数，同时产生新的变量空间。pause 指令之前的那一级的变量在这一级可引用，通过任何的输出语句可以将这些变量的值显示出来，因而可以达到检测函数内局部变量的目的。

虽然 pause 指令可以在新的窗口界面内对源程序内的变量进行操作，但是不能将高层空间内赋值的变量带到原调用函数的工作空间，不能改动原来函数中局部变量的值。如果用户需要将高层空间中变量的值来改变底层空间中局部变量的值，可以通过 return 或 resume 指令来实现。

3. return/resume

在使用 pause 指令后，要回到上一级，用指令“return”或“resume”二者是等价的。

return 指令还可以“[...]=return(...)”的形式将高一级的变量值带回到低一级。

```
-->a = 1;

-->pause                    //跳到第一级

-1->a                       //可以读取低一级空间的变量
a   =

    1.

-1->pause                   //跳到第二级

-2->a = 2;

-2->a = return(a)           //在第二级给变量 a 赋值

-1->a
a   =

    2.

-1->return

-->a
a   =

    1.
```

pause 指令在编程调试的时候非常有用。在程序运行过程中，可以直接通过键盘的“ctrl-c”实现暂停，以中止那些无限循环。“abort”指令中止暂停状态，退出调试。

在程序调试的过程中，用户如果需要执行程序中的部分代码，可以在 Scilab text editor 编辑器窗口中选中需要执行的代码，然后单击“执行”菜单，选择“执行选择部分”就可以实现对程序中的部分代码的执行。

4.7 错误和异常处理

在编写或调试程序时无论有多么仔细，应用程序的运行都不会像期望中的那

样一帆风顺，或多或少都会出现一些错误，从而产生一些意想不到的结果，称之为出现异常。在程序中有一些专门用于处理这些错误的代码，称为异常的捕获和处理。程序中有了异常的捕获和处理代码，无疑是一件令人高兴的事情。

在 Scilab 中可以通过 try-catch 模块来进行异常的捕获和处理，下面这个例子显示了如何捕获和处理异常。

```
function testtry(fc)
try
  deff("[r] = func(x)","r = " + fc);
catch
  disp("在函数的定义过程中出现语法错误!");
  //disp("error!");
  ferror = %T; return;
end
endfunction
```

从上面的例子可以看出，异常可分为两个模块，第一个有 try 开始，称为异常捕获模块；第二个模块由 catch 开始，称为异常处理模块，即抛出异常。

在上面的例子中，如果字符型变量 fc 不是 Scilab 的函数表达式，即不满足 Scilab 函数的定义规则，那么 try-catch 模块将会捕获该异常，并显示 catch 中的字符串。我们将此函数保存后执行，运行结果如下：

```
-->exec('/home/scilab/trytest.sci', -1)

-->testtry('2^x(1')

在函数的定义过程中出现语法错误!
```

try-catch 模块可以嵌套使用，如下面的例子所示。

```
function nestedtry(a,b)
disp("START")
mprintf("\ta is %s\t\tb is %s\n",string(a),string(b))
try
  disp("try 1")
  try
    disp("try 2")
    z = a + 1;
  catch
    disp("catch 2")
    t = b + 1;
  end
```

```
  disp("after try 2")
catch
  disp("catch 1")
end
disp("after try 1 - THE END")
endfunction
```

在调试过程中我们将 a 和 b 设置成字符，此时抛出异常的运行结果如下：

```
-->nestedtry(1,1)

START
  a is 1   b is 1

try 1

try 2

after try 2

after try 1 - THE END

-->nestedtry("a string",1)

START
  a is a string   b is 1

try 1

try 2

catch 2

after try 2

after try 1 - THE END

-->nestedtry(1,"a string")

START
```

```
  a is 1   b is a string

try 1

try 2

after try 2

after try 1 - THE END

-->nestedtry("a string","a string")

START
  a is a string   b is a string

try 1

try 2

catch 2

catch 1

after try 1 - THE END
```

出此，我们可以通过抛出异常的结果不同来区分 a 和 b 到底哪一个是字符串。

虽然 try-catch 模块可以用来捕获和抛出异常，但是在更多情况下，仅仅需要报告错误和约束程序的运行，此时可以使用 error 函数。

使用 error 函数可以向用户报告错误，通过错误信息的报告来处理错误。默认情形下 error 函数将停止程序进程的执行，并给出错误的提示信息。

一些预定义的错误信息需要一个参数，用户可以参见 Scilab 帮助文档中的 error_table 表中所列出的错误信息，其中的一些错误信息被 Scilab 自身的分析器错误或相关的内建错误所使用。例如输入的矩阵行和列的维数不一致时，Scilab 会提示如下错误：

```
-->A=[1 2 3;5 6]
                ! --error 6
不一致的行维数/列维数。
```

有一些更为普通的错误可以在编写程序的过程中直接使用，特别是在程序的

输入有特别要求时，可以通过错误信息的提示使得用户能够正确地运行程序。主要的调用格式有如下 3 种：

error(message [,n])：在错误信息中打印字符串，与错误信息关联的编号为 10000。

error(n)：在错误信息中打印字符串，与此错误消息关联的编号由 n 给定，这个数字应该比 10000 更大。

error(n,pos)：打印预定义的错误信息，与此错误消息关联的编号由 n 给定。

例如，在程序中要求输入参数 n 必须要大于 5，可以使用如下代码实现：

```
if n<=5
  error('n必须大于5!')
end
```

5 Scilab 绘图

在科学计算领域，人们很难直接从一大堆原始的离散数据中理解它们的含义，而数据图形却能使人快捷、直观地理解数据的许多内在本质。完备的图形功能使计算结果可视化，是 Scilab 的重要特点之一。用图表和图形来表示数据的技术称为数据可视化。Scilab 具有良好的数据可视化功能，可以将数据用二维或三维图形表现出来。

5.1 图像窗口介绍

在 Scilab 中能用多个图像窗口显示图形，Scilab 自动将其命名为图像窗口数目 a，其中 a 是窗口的编号。在任何时刻只能有一个窗口被激活。在 Scilab 的运行主窗口中。用菜单中的“Graphic window x”项来管理图像窗口（x 表示当前激活窗口的编号）。利用该菜单能新建、弹起或删除第 x 窗口。用户可以直接建立任意标号的窗口。如果必要，则绘图指令会自动建立一个新的图像窗口，如图 5.1 所示。

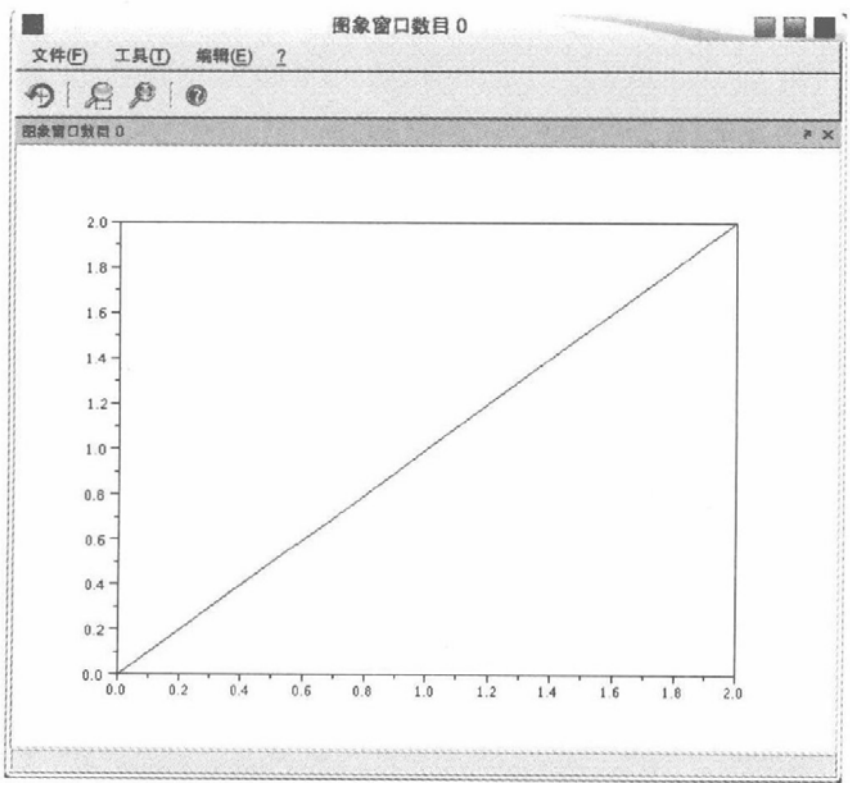


图 5.1 Scilab 图像窗口

在 Scilab 图像窗口中，其菜单栏共包含 4 个菜单按钮：

(1) 文件菜单，包括：

新 Figure　用于打开一个新的 Scilab 图像窗口。

载入　将一个图形文件载入 Scilab 图像窗口。

保存　保存当前 Scilab 图像窗口中的图形文件，默认的文件格式为 .scg。

导出到　输出当前 Scilab 图像窗口中的图形文件，可以以图片格式保存窗口中的图形，所支持的格式包括：BMP、GIF、JPEG、PNG、PDF 等。

复制到剪切板　将当前图形窗口中的图形文件复制到剪切板，以便粘贴到其他文件或文档中。

页面设置　用于设置当前图形窗口的纸张大小、方向、页边距等，便于图形的打印。

打印　根据页面设计的效果，对当前图像窗口中的图形进行打印。

关闭　关闭该图像窗口。

图像窗口中的文件菜单如图 5.2 所示。

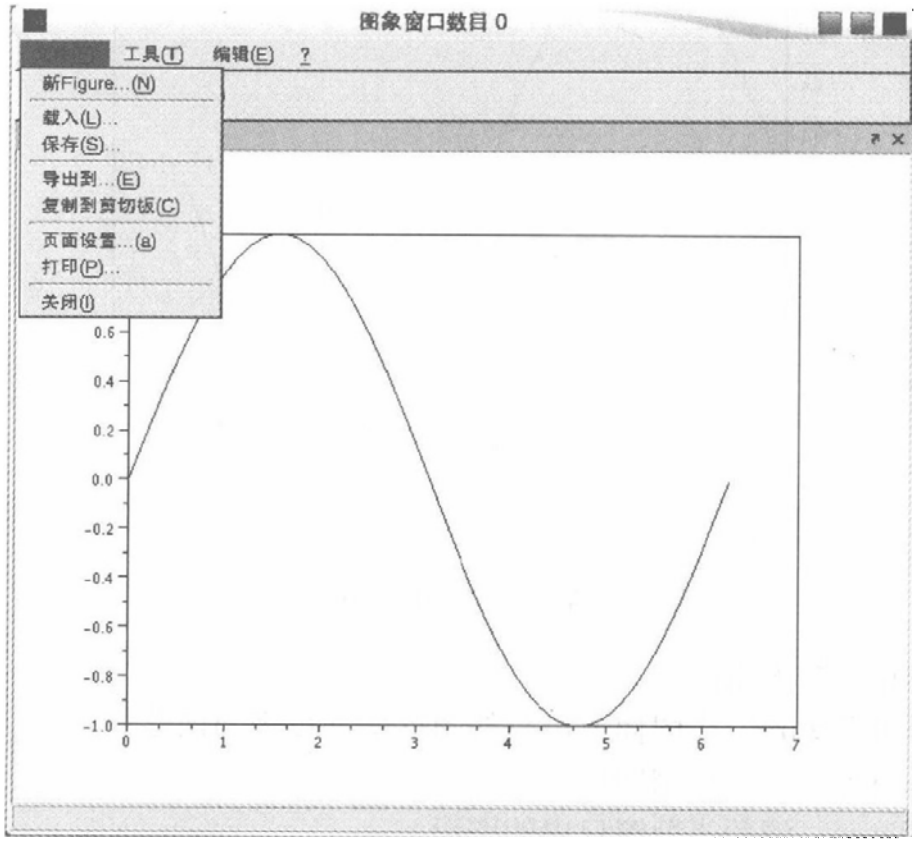


图 5.2　Scilab 图像窗口中的文件菜单

(2) 工具菜单,包括:

显示/隐藏工具栏　可以用来显示或隐藏工具栏。

缩放　按比例放大或缩小图像窗口中的图形文件,与工具栏中的图标功能相同。

取消缩放　取消前次放大或缩小,与工具栏中的图标功能相同。

2D/3D 旋转　2 维或 3 维旋转,与工具栏中的图标功能相同。

图像窗口中的工具菜单如图 5.3 所示。

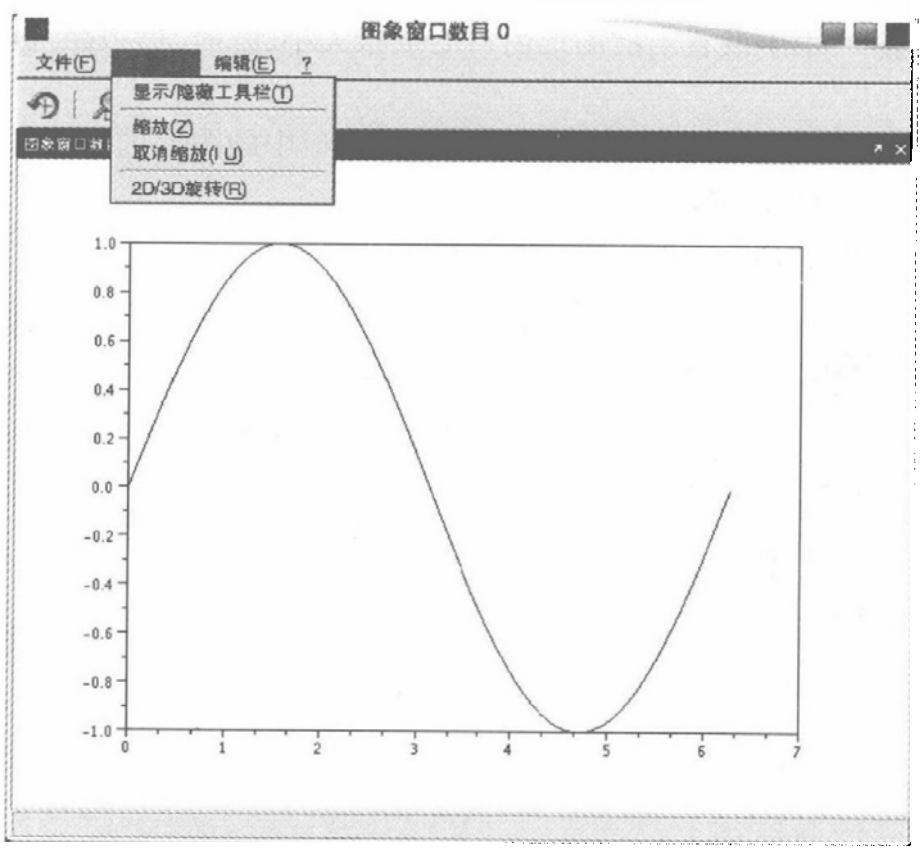


图 5.3　Scilab 图像窗口中的工具菜单

(3) 编辑菜单,包括:

设为当前 Figure　将图像窗口中的图形文件设为当前绘图。

重绘 Figure　重新绘制图形。

清除 Figure　清楚当前窗口内的图形。

Figure 属性　当前绘图的属性,点击以后会弹出如图 5.4 所示的窗口。

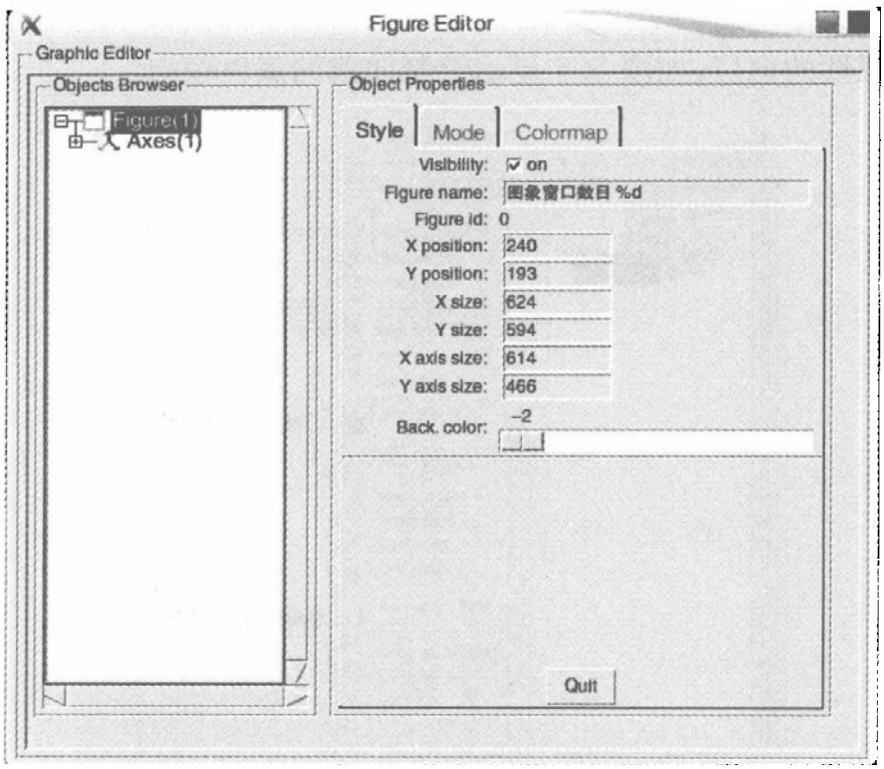


图 5.4 当前绘图的属性窗口

坐标轴属性 当前绘图的坐标轴属性，点击以后会弹出如图 5.5 所示的窗口。

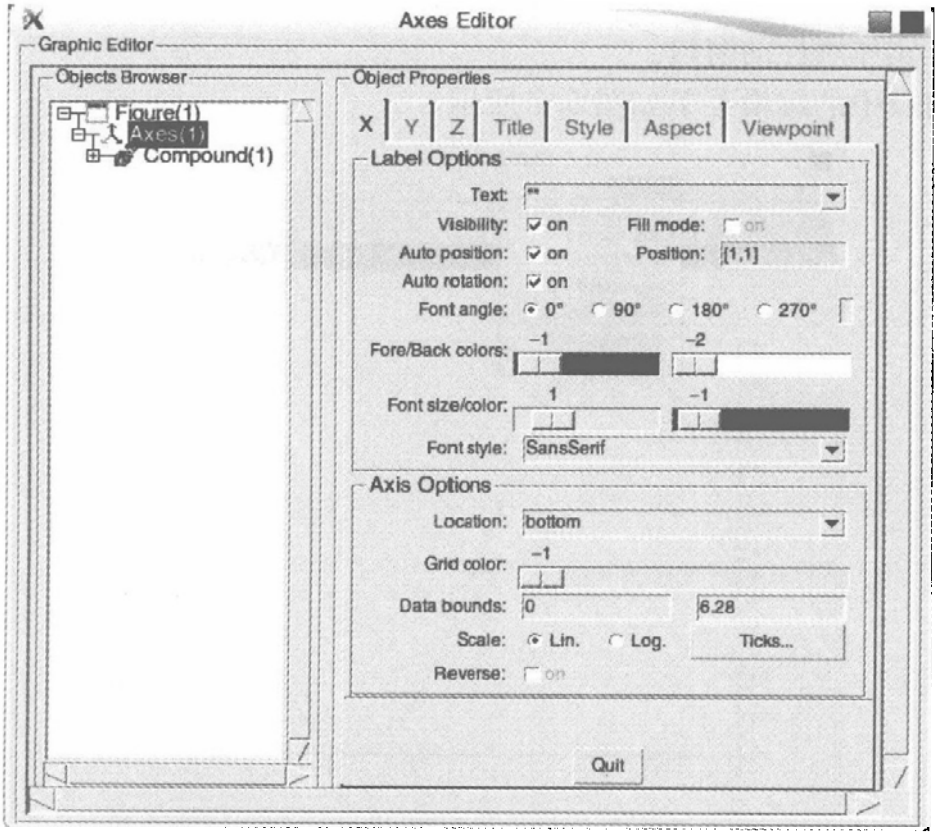


图 5.5 当前绘图的坐标轴属性窗口

开始选择实体　启动实体捡拾器，单击以后在绘图窗口中选中实体时，会弹出关于该实体属性的窗口，如图 5.6 显示绘制曲线的属性窗口。

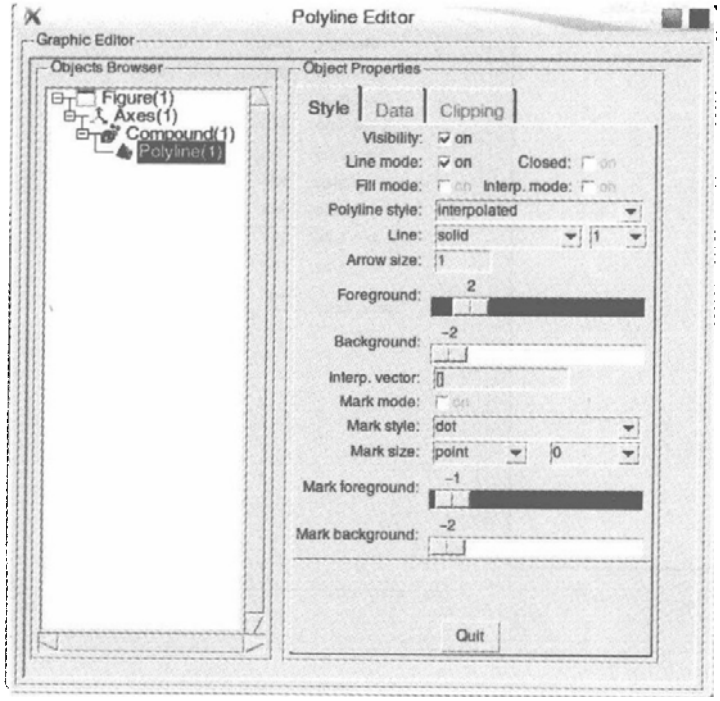


图 5.6　绘制曲线的属性窗口

停止选择实体　停止实体捡拾器，单击以后在绘图窗口中选中实体时，不再会弹出关于该实体属性的窗口。

图像窗口中的编辑菜单如图 5.7 所示。

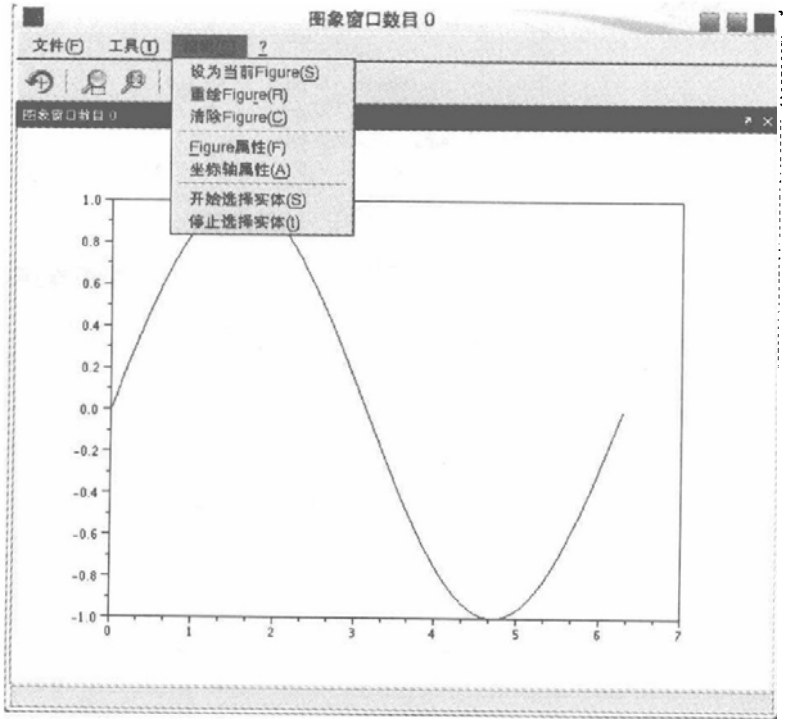


图 5.7　Scilab 图像窗口中的编辑菜单

(4) ?菜单,包括:

Scilab 帮助 Scilab 帮助菜单,与工具栏中的图标功能相同。

关于 Scilab 显示 Scilab 的版权信息等。

图像窗口中的帮助菜单如图 5.8 所示。

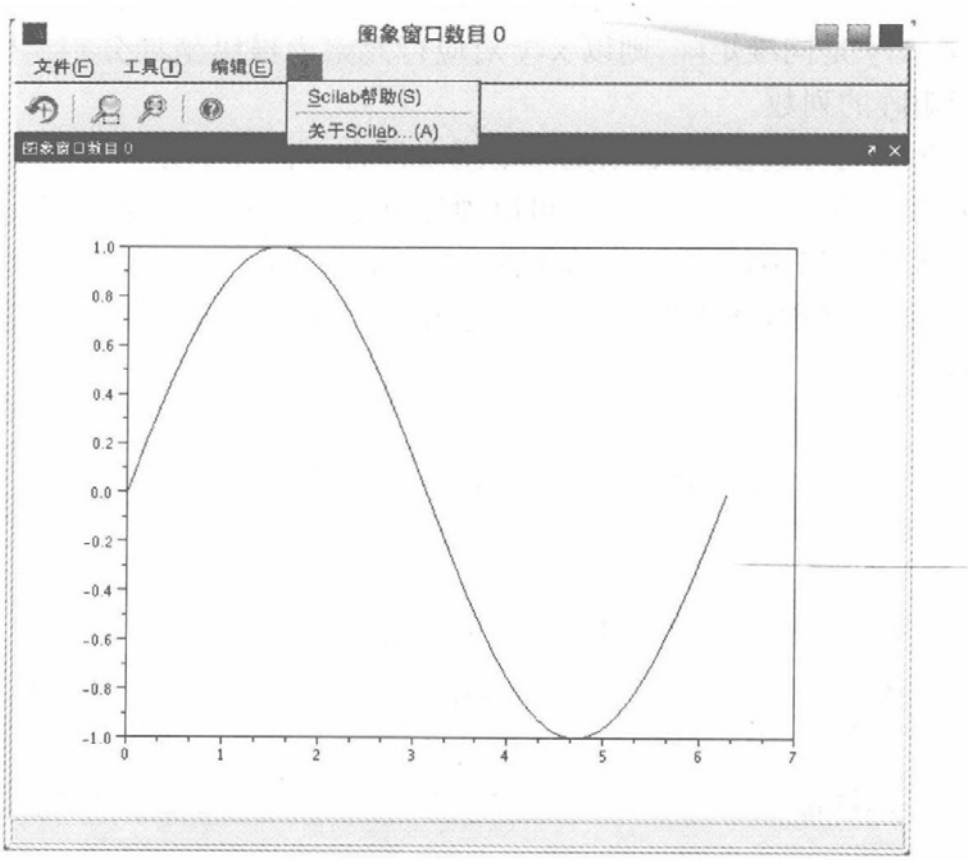


图 5.8 Scilab 图像窗口中的帮助菜单

5.2 二维图形的绘制

下面介绍 Scilab 中二维图形的绘制方法,主要通过两个指令函数来实现:plot 和 plot2di。

5.2.1 plot 指令

plot 是最基本的二维图形绘制指令,它属于 Scilab 的内部函数。下面介绍 plot 指令绘制二维图形时的基本格式:

1. plot(y,<LineSpec>,<GlobalProperty>)

若 y 为向量,则以 y 元素值为纵坐标,以相应元素下标为横坐标值绘制连线图;若 y 为实数矩阵,则按行绘制每行元素值并对应列下标的连线图,图中的曲线

数等于 y 阵的行数。

2. plot(x,y,<LineSpec>,<GlobalProperty>)

若 x 和 y 是两个向量，则其中 y 是 x 的函数；在 x 没有给定的情况下，用向量(1 : size(y,' * '))代替横坐标值；若 y 是一个实数矩阵，则按行绘制每行元素的连线图；若 x,y 是同维矩阵，则以 x,y 对应行元素为横纵坐标分别绘制曲线，曲线条数等于矩阵的列数。

LineSpec 为可选参数，必须是一个用来作为一种快捷方式指定一个画线方式的字符串，对于每一个 y 或{x,y}，可以预先设定 LineSpec 参数。LineSpec 参数指定线条样式、标记符和所绘制线的颜色。GlobalProperty 也是可选参数，表示一对属性——{属性名称，属性值}，定义了适用于所有由该 plot 指令创建曲线的全局对象的属性。

例 5.1 单向量输入格式，画折线图。

```
-->y=[8,5,9,7,0,6,3];
-->plot(y)
```

执行以上指令后，得到如图 5.9 所示的图像窗口。

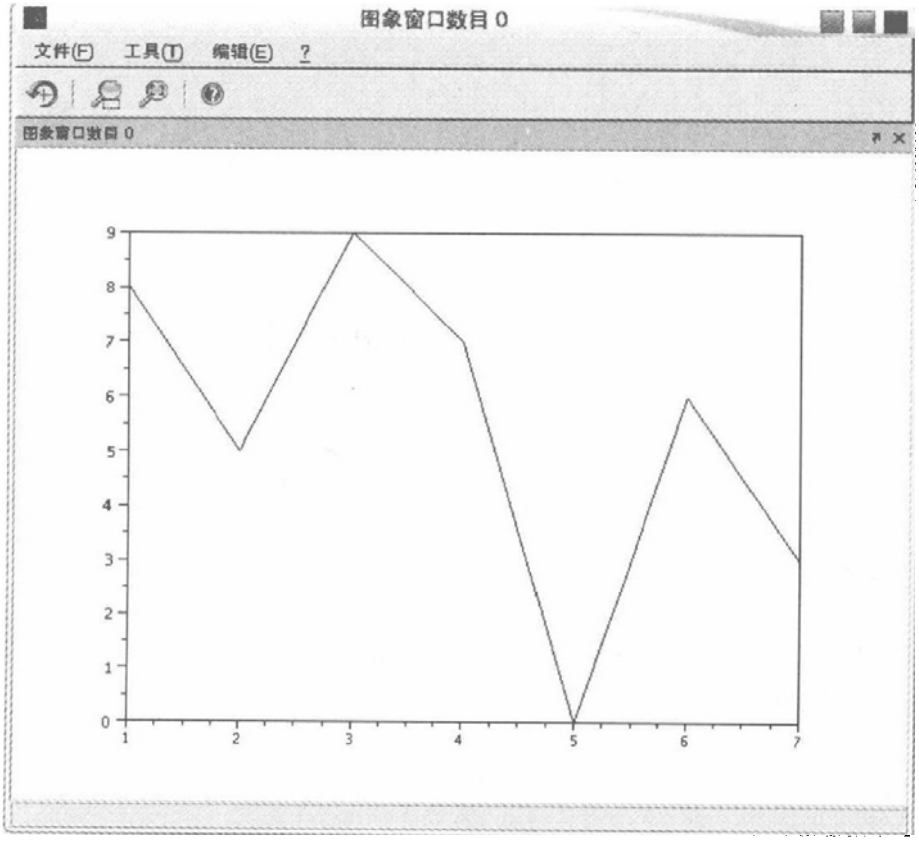


图 5.9 利用 plot(y)绘图

例 5.2 双向量输入格式，画曲线。

```
-->x = 0 : 0.01 : 2 * %pi;
-->plot(x, sin(x))
```

说明：本例第一条赋值语句使 x 成为行向量，这是最常见的绘制图形生成方式。可以用 plot(sin(x))指令代替 plot(x,sin(x))指令，效果完全一样。

执行以上指令后，得到如图 5.10 所示的图形窗口。

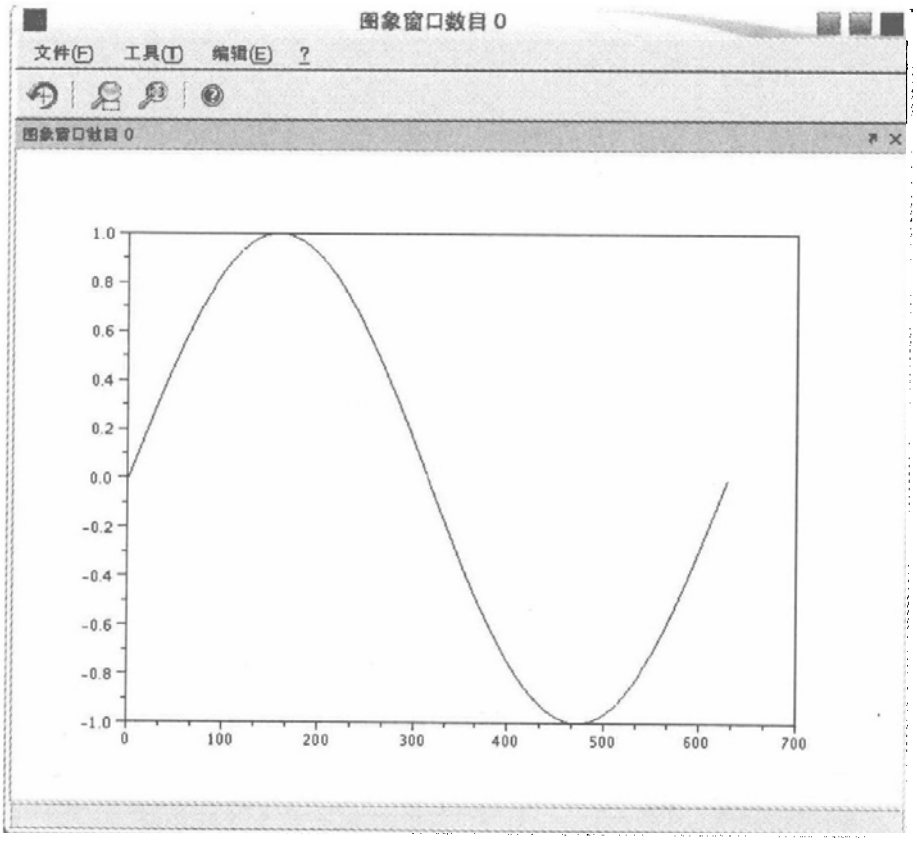


图 5.10 利用 plot(x,y)绘图

例 5.3 在输入量一个是向量，一个是矩阵的情况下，以相同横坐标画多根直线。

```
-->a = 0 : 0.01 : 10;
-->plot(a,[2 * a - 1; - 3 * a + 5])
```

说明[2 * a－1；－3 * a＋5]组成一个矩阵。注意 plot(y)与 plot(x,y)的区别，plot(y)是通过坐标为(i,y_i)的点连线确定，而 plot(x,y)是通过坐标为(x_i,y_i)的点连线确定。在 y 为矩阵时，x 必须与 y 为同阶矩阵，此时所画曲线系由 x,y 的对应列向量所画曲线的连线。

执行以上指令后，得到如图 5.11 所示的图像窗口。

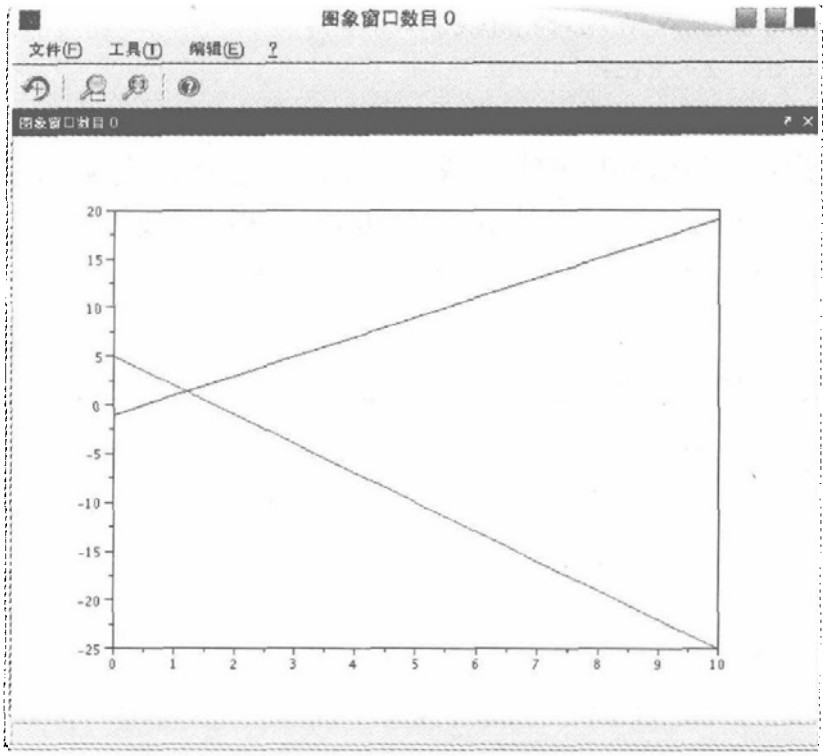


图 5.11 利用 plot(x,y)绘制多根直线

例 5.4 利用不同标识绘制图像。

```
-->t = 0 : %pi/20 : 2 * %pi;
-->plot(t,sin(t),'ro-.',t,cos(t),'b+',t,abs(sin(t)),'--mo')
```

执行以上指令后,得到如图 5.12 所示的图像窗口。

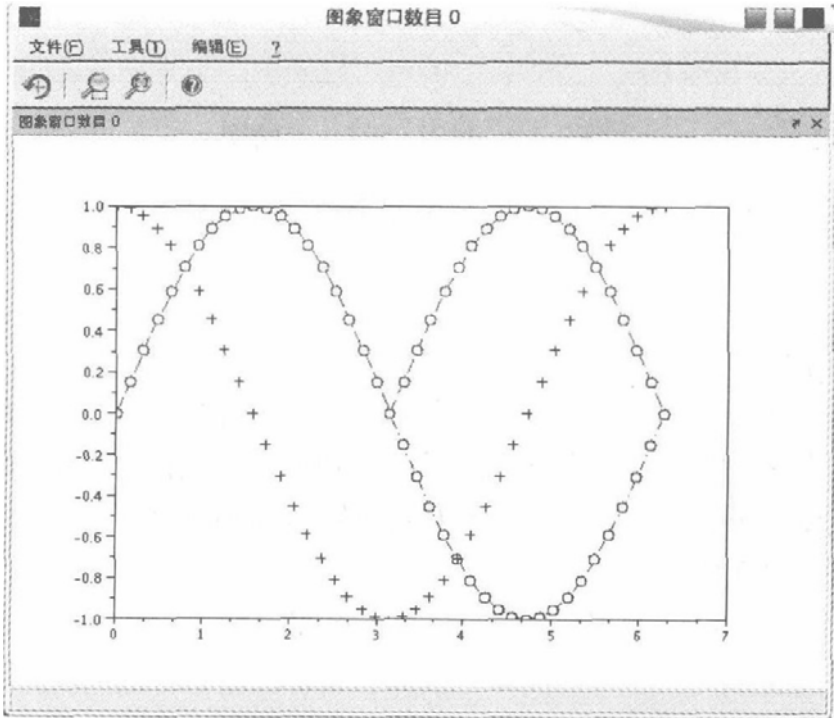


图 5.12 利用不同标识绘图

5.2.2 plot2d(i)指令

1. plot2d([x,]y)

x 和 y 可以是两个实矩阵或向量。如果 y 是一个向量，则 x 必须是一个具有相同元素个数的向量。在 x 没有给定的情况下，用向量(1:size(y,'*'))代替 x；如果 y 是一个矩阵，x 可以是一个与 y 行数相同的向量，或与 y 具有相同阶数的矩阵，y 的每列根据相关的 x 绘制。

例 5.5 绘制 sin(x)在[0,2π]上的图像。

```
--> x=[0:0.1:2*%pi]';
--> plot2d(x,sin(x));
```

因为 x 参数是可选项，可以用 plot2d(sin(x))指令代替 plot2d(x,sin(x))指令，效果完全一样。

执行以上指令后，得到如图 5.13 所示的图像窗口。

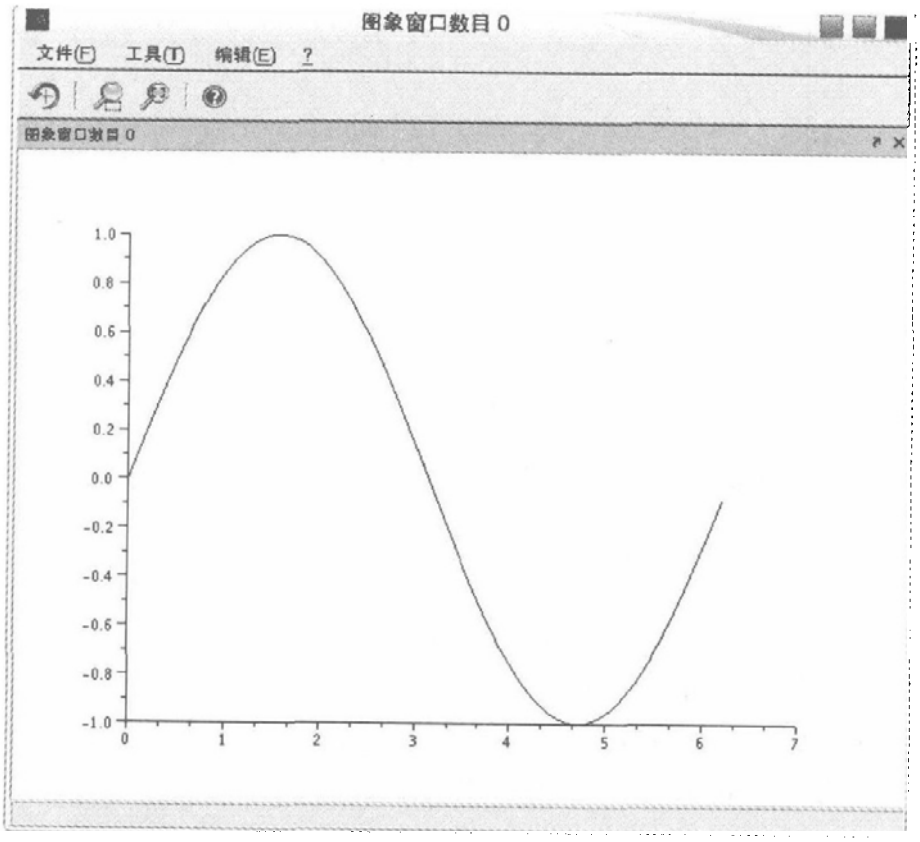


图 5.13 利用 plot2d 指令绘图

例 5.6 在同一坐标平面内分别在区间[0,π]绘制 sin(x)，在[π,2π]上绘制 cos(x)+1 的图像。

```
-->x=0:0.01:%pi;
-->x1=%pi:0.01:2*%pi;
```

```
-->z=[x;x1]';
-->y=[sin(x);cos(x1)]';
-->plot2d(z,y)
```

执行以上指令后，得到如图 5.14 所示的图像窗口。

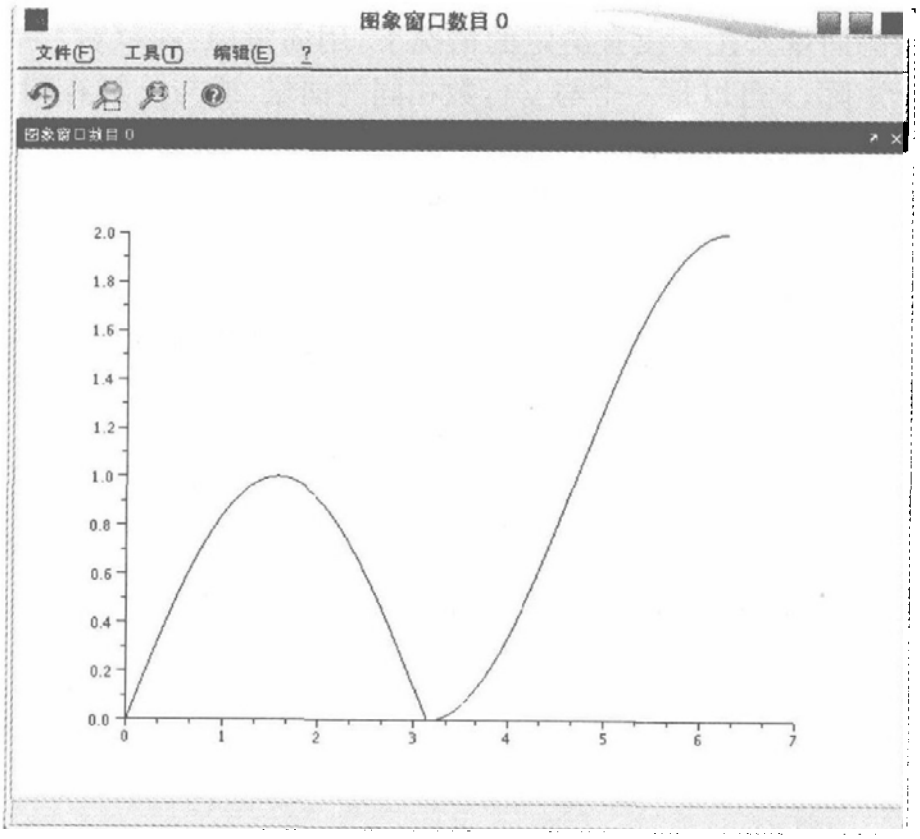


图 5.14 利用 plot2d 指令绘制多条曲线

利用本例的方法可以绘制分段函数的图像。

2. plot2d([x,]y,<opt_args>)

x 和 y 可以是两个实向量或实矩阵。如果 y 是一个实向量，x 就必须是维数与 y 相同的实向量。若 x 缺省，Scilab 会认为 x 是向量[1:size(y,'*')]。若 y 是一个实矩阵，则 x 可以是一个维数与 y 的行数相同的实向量或者是与 y 同阶的矩阵。将 x 与 y 的对应列上的相应分量作为点的坐标，连线可画出 m 条曲线(m 为矩阵 y 的列数)。

可缺省参数<，opt_args>是一个参数声明序列，其形式如：key1＝value1，key2＝value2，…，其中 key1，key2，…可能是下列参数之一。

style 为每条曲线线型设置向量，该向量每一分量都应取整数值。第 i 条曲线的线型由 style(i)确定。由此可知，此项一旦选定，其分量应该和 y 的列数一样(在 y 是矩阵的情形下)，即和所要画的曲线条数一样。如果 style(i)是正的，则第 i 条

曲线作为平面曲线其颜色索引项就由 style(i)确定(参看 getcolor 指令)。应该指出的是,曲线的线型与厚度还可以通过多线性实体性质(参看 polyline properties)进行设置。在给定曲线上每相邻两点之间用逐段线性插值来画出该曲线。如果 style(i)非正,则给定曲线上的点用 abs(style(i))标识画出。

logflags 选项用来设置坐标轴的刻度类型是线性刻度还是对数刻度。与之相关的值应该为下列四种字符串中的一个:"nn","nl","ln"及"ll"。其中"l"表示对数刻度,"n"则表示通常的线性刻度或标准刻度。

rect 选项用来设置为画出一平面图形所需要的最小界限,与之相联系的有四个元素的实向量:[xmin,ymin,xmax,ymax]。xmin 和 xmax 定义横坐标的界限,而 ymin 和 ymax 则定义纵坐标的界限。这个选项可以和 frameflag 选项一起使用,用以表明坐标轴的界限如何从 rect 选项中导出。如果 frameflag 缺省,则 Scilab 认为此时 frameflag=2。

frameflag 选项用 0~5 的任一数来控制实际坐标范围的计算,该范围可能比最小所需范围来得大些。

frameflag=0:使用前一制图所设定的坐标范围或缺省设置来制图,因而无须进行任何计算。

frameflag=1:选择此选项坐标范围比在 rect 选项中设定的略大。

frameflag=2:坐标范围从 x 和 y 数据的 min/max 中计算得出。

frameflag=3:坐标的实际范围就是 rect 中给定的范围,并按等比例放大。

frameflag=4:坐标的实际范围由 x 和 y 数据的 min/max 计算给出,且按等比例放大。

frameflag=5:实际范围就是 rect 选项中给定的范围,但被放大以得到比较美观的坐标轴标号。

frameflag=6:实际范围从 x 和 y 数据的 min/max 中计算得出,但被放大后得到较为美观的坐标轴标号。

frameflag=7:与 frameflag=1 相同,但以前画的图将使用新的比例被重画,通常用此种选项将当前图加于以前画的图内。

frameflag=8:与 frameflag=2 相同,但以前画的图将使用新的比例被重画,通常用此种选项将当前图加于以前画的图内。

frameflag=9:与 frameflag=8 相同,但实际范围被放大以后得到比较美观的坐标轴标号,这是默认值。

rect 和 frameflag 主要用来设定坐标系的范围(或边界),具有类似功能或可以设定坐标系范围的还有 axes properties。

axesflag 选项用 0~5 的一个整数来设定坐轴在图中位置或显示方式。

axesflag=0：在所画图上不显示坐标轴（axes_visible=["off" "off"];box="off"）。

axesflag=1：两坐标轴均被画出，但y轴被画在图的左边（或y轴在图的左边显示）（axes_visible=["on" "on"];box="on",y_location="left"）。

axesflag=2：图被一没有刻度的方框环绕（axes_visible=["off" "off"];box="on"）。

axesflag=3：两坐标轴均被画出，但y轴被画在图的右边（或y轴在图的右边显示）（axes_visible=["on" "on"];box="off",y_location="right"）。

axesflag=4：两坐标轴被画在图的中央（axes_visible=["on" "on"]; box="off", x_location="middle", y_location="middle"）。

axesflag=5：将两坐标轴画在点(0,0)处，若点(0,0)不在此图的构架范围之内，则两坐标轴不出现在所画图上（axes_visible=["on" "on"];box="on",x_location="middle",y_location= "middle"）。

axesflag=9：两坐标轴均被画出，y轴被画在图的左边（或y轴在图的左边显示），这是默认值。

除了axesflag选项可以设定坐标轴外，还可用axes properties进行坐标轴的相应设定。

下面介绍坐标轴上的标识或刻度的设定。

nax：当axesflag=1时，可用此选项来设定坐标轴上的标识或刻度。与此选项相对应的是有四个整数元素的实向量：[nx,Nx,ny,Ny]。Nx(Ny)给定x(y)轴上的主刻度，nx(ny)则给出了x(y)轴上的次级刻度。所谓刻度，类似于学生直尺上的厘米与毫米刻度。在缺省的情况下（即没有设定），nax选项认为axesflag已设定为1。

接着再介绍怎样给曲线设定说明文字或标识。

leg：这个选项用来设定图中每条曲线的标示或说明，它必须采用形如"leg1@leg2@ leg3@…"的字符串形式，其中leg1，leg2…分别是第一条曲线、第二条曲线……的标题或文字说明。其默认形式为" "，曲线的标题被画在x轴下面。这个选项不十分灵活，用legends设定标题说明更为可取。

例5.7 利用plot2d指令绘制二维图形。

```
-->x = -1 : 0.01 : 1;
-->y = sqrt(1 - x^2)';
-->z = -y;
-->plot2d(x,[y,z],logflag = "nn",frameflag = 3,rect = [-2, -2,2,2])
```

执行以上指令后，得到如图5.15所示的图像窗口。

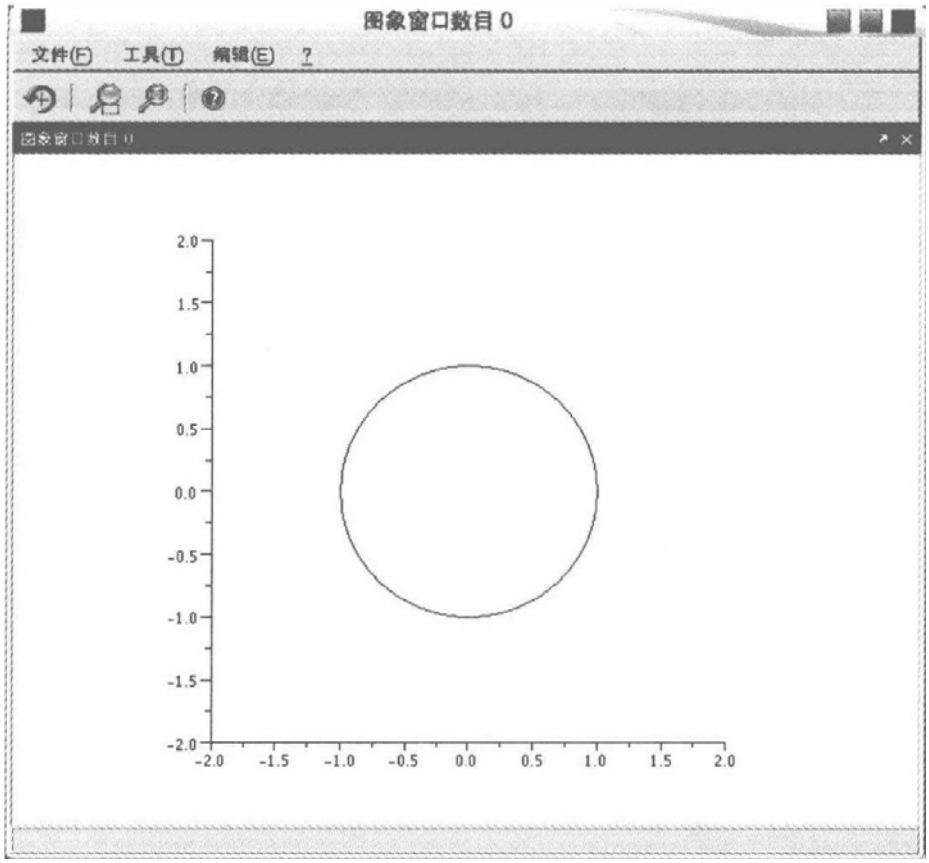


图 5.15 利用 plot 2d 指令绘图

例 5.8 同时绘制多条曲线。

```
-->x = 0 : 0.01 : 2 * %pi;
-->plot2d(x,[sin(x)' sin(2 * x)' sin(3 * x)'],...
style = [1,2,3],leg = "sinx@sin2x@sin3x",nax = [2,10,2,10],rect = [0, - 2,2 * %pi,
2]);
```

从例子可以看到，style 设置了曲线的颜色，rect 设置了绘图的范围，nax 给定了 x 轴与 y 轴的刻度，leg 设置了曲线的文字标题。

执行以上指令后，得到如图 5.16 所示的图像窗口。

例 5.9 用不同标度在同一坐标内绘制曲线。

利用 Scilab 提供的 drawaxis 指令可以实现在同一坐标内用不同标度来绘制曲线，类似于 Matlab 中的 plotyy 函数(关于 drawaxis 指令的具体用法可以参见 Scilab 帮助文档)。

```
x = 0 : %pi/360 : 2 * %pi;
y1 = exp( - 0.5 * x). * cos(2 * x);
y2 = 10 * exp( - 1.1 * x);
plot2d(x,y1,2,"020");
plot2d(x, - 1 + y2/5,5,"020");    //绘制 y2 时通过变换 - 1 + y2/5,将 y2 绘制到 y1 的范围内
```

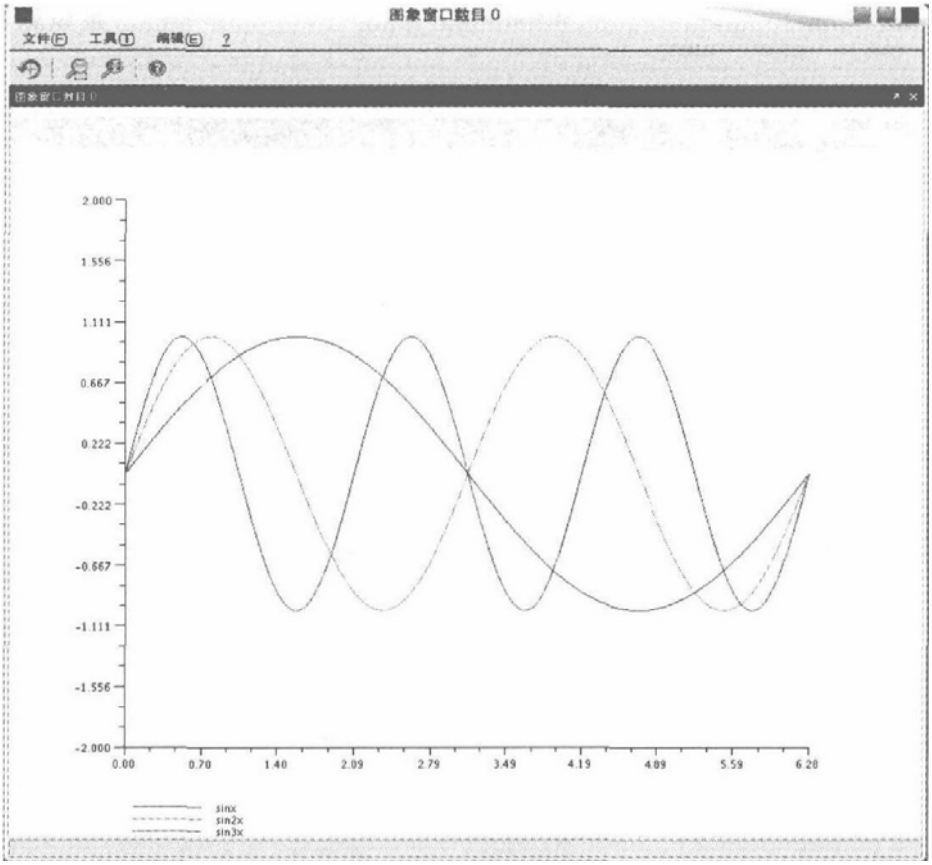


图 5.16 利用 plot 2d 指令同时绘制多条曲线

```
drawaxis(dir = "l",x = 0,y = -1 : 0.2 : 1,val = string(-1 : 0.2 : 1),fontsize = 1,
textcolor = 2,ticscolor = 2)
//绘制左边的 y 轴,刻度从 -1 到 1
drawaxis(dir = "r",x = 7,y = -1 : 0.2 : 1,val = string(0 : 1 : 10),fontsize = 1,text-
color = 5,ticscolor = 5);
//绘制右边的 y 轴,刻度从 1 到 10
drawaxis(dir = "d",x = 0 : 1 : 7,y = -1,val = string(0 : 1 : 7),fontsize = 1,textcolor
= 0,ticscolor = 0);
//绘制 x 轴,刻度从 0 到 7
legend(["exp(-0.5 * x). * cos(2 * x)" "10 * exp(-1.1 * x)"],pos = 1)
```

由以上代码所绘制的曲线图如图 5.17 所示。

较常用的二维绘图指令还有 plot2di,它包含一组绘图指令:

plot2d1:比 plot2d 多一个形式参数,其余与 plot2d 相同,可同时绘制多条曲线;

plot2d2:用分段常量方式绘制图形;

plot2d3:绘制柱形图;

plot2d4:用箭头绘图。

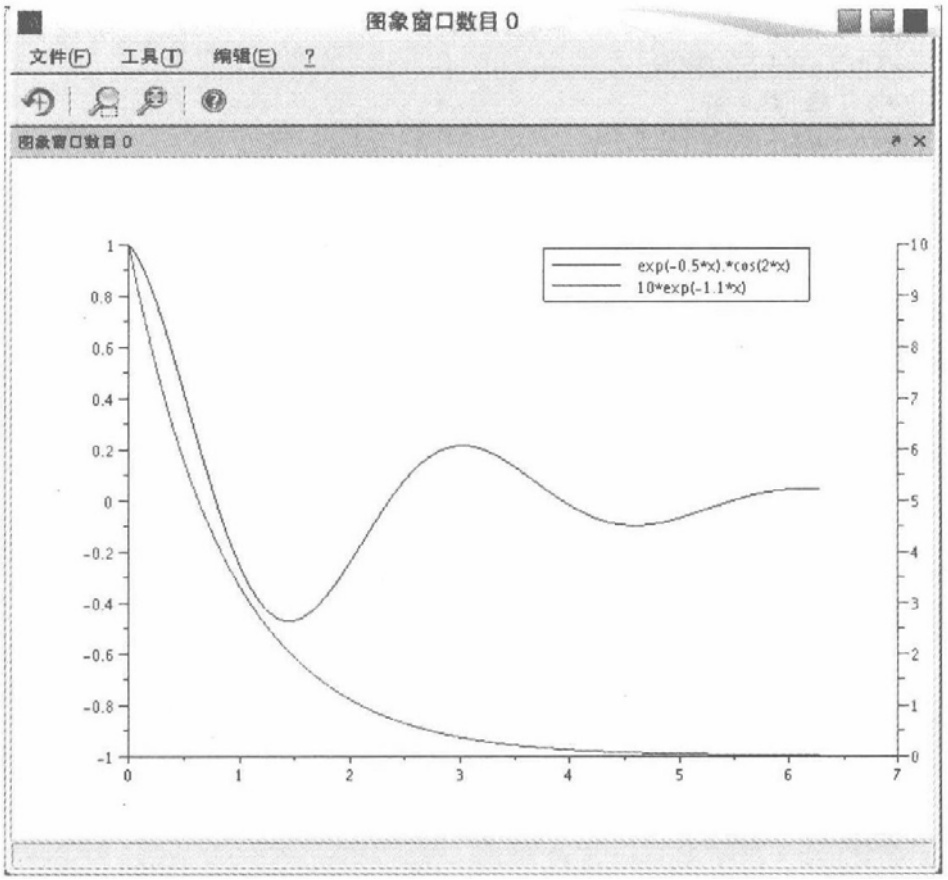


图 5.17 在同一坐标内绘制不同标度的曲线

例 5.10 用 plot2d2 绘制函数 y=cos2x−sinx 在[0,4π]上的图像。

```
-->x = 0 : 0.3 : 4 * %pi;
-->plot2d2(x,cos(2 * x) - sin(x))
```

如图 5.18 所示,plot2d2 绘出的是阶梯图,类似于 Matlab 中的 stairs 函数。

例 5.11 用 plot2d3 绘制函数 y=x^0.8 * e^−x 在[0,4]上的图像。

```
x = 0 : 0.1 : 4;
y = (x.^0.8). * exp( - x);
plot2d3(x,y)
plot2d(x,y,style = - 9)
```

在后面加上 plot2d(x,y,style=−9),是在原图的上面加上了空心圆点,就可以得到类似于 Matlab 中的函数 stem 所绘制的针状图,如图 5.19 所示。

例 5.12 用 plot2d4 绘制函数 y=x * sinx 在[0,2π]上的图像。

```
-->x = 0 : 0.2 : 4 * %pi;
-->plot2d4(x,x. * sin(x))
```

对于指令 plot2di(i=1,2,3,4),同样可以像 plot2d 一样,指定 style,rect,frameflag,axesflag 等参数。

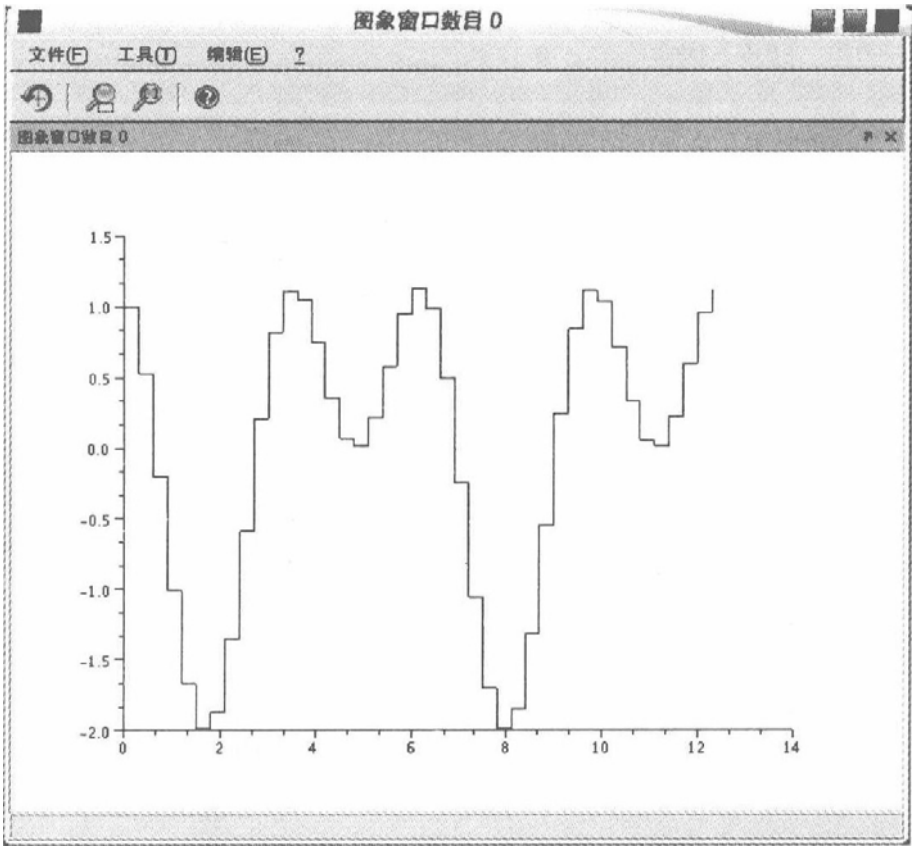


图 5.18 利用 plot2d2 指令绘制阶梯图

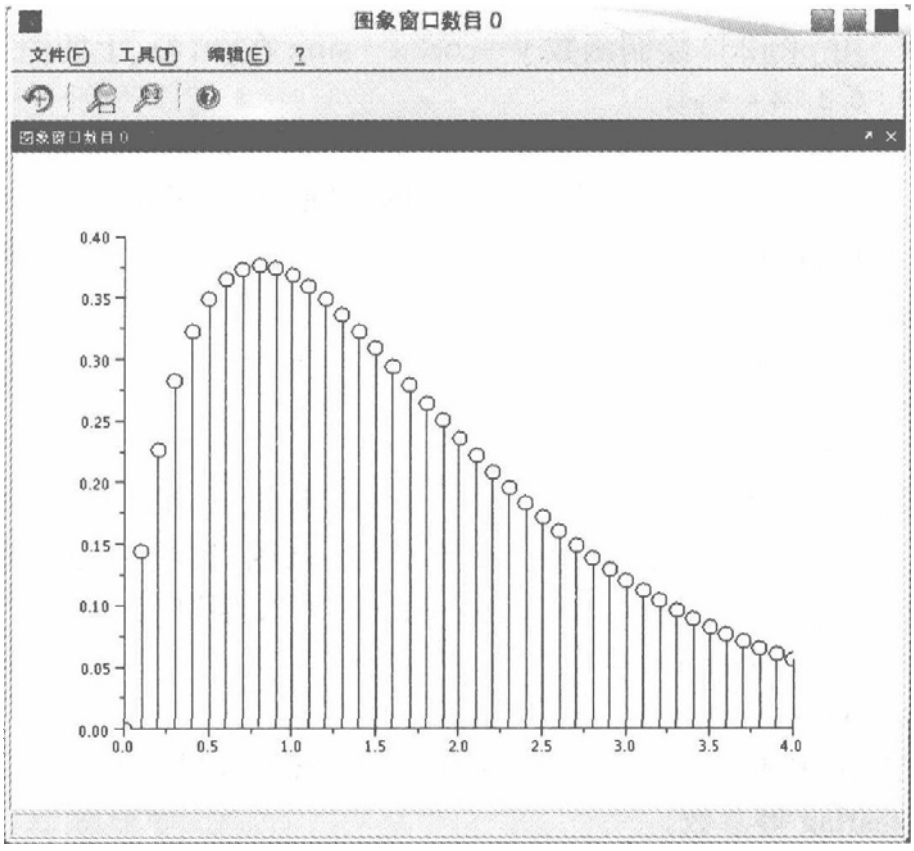


图 5.19 利用 plot2d3 与 plot2d 指令绘制针状图

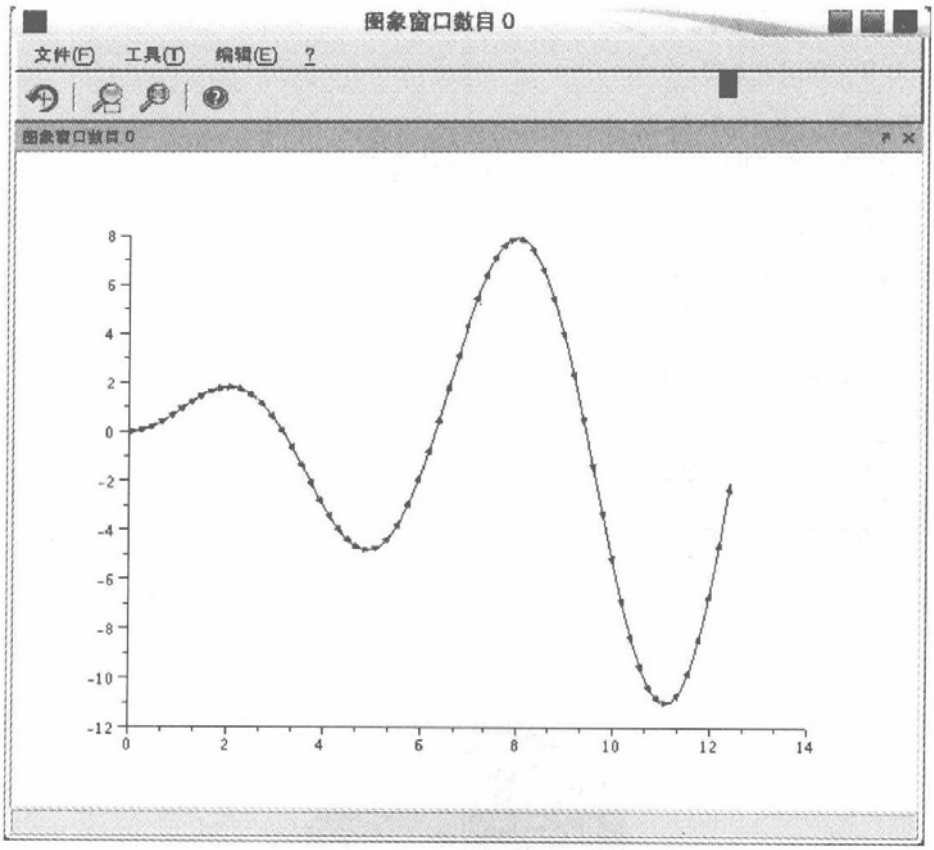


图 5.20 利用 plot2d4 指令绘图

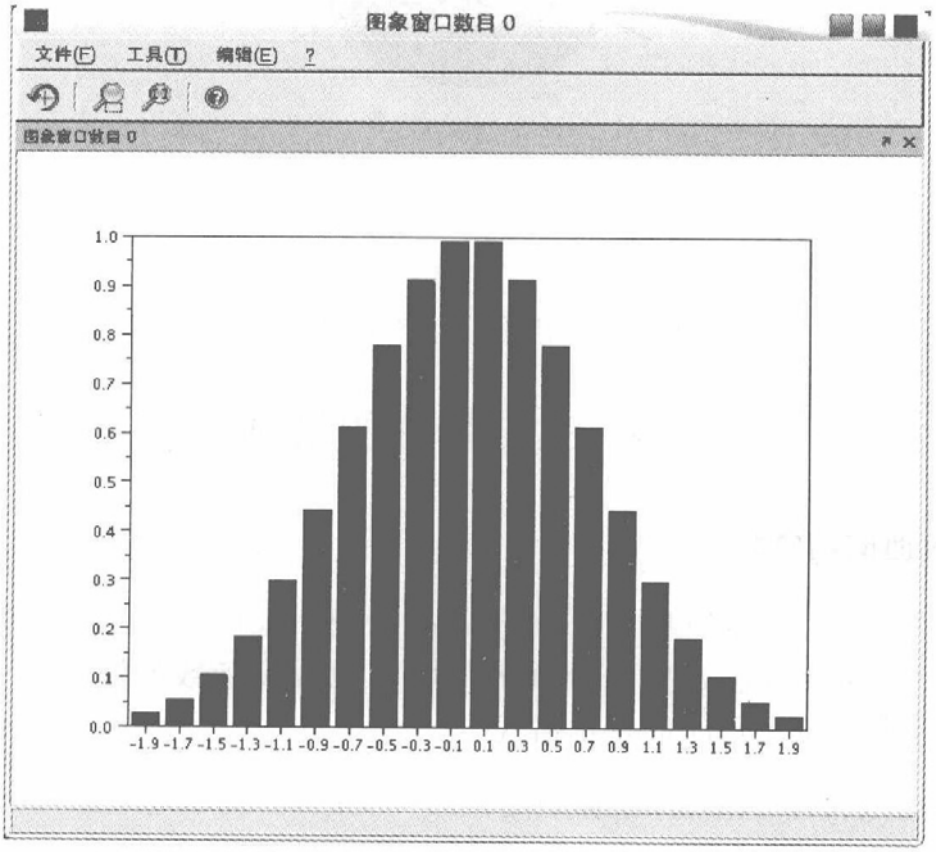


图 5.21 利用 bar 指令绘制条形图

例 5.13 绘制条形图。

```
-->x = -1.9 : 0.2 : 1.9;
-->bar (x,exp( -x. * x));
```

执行以上指令，得到如图 5.21 所示的图像窗口。

例 5.14 绘制饼图。

```
-->x = [28,65,19,86,59,97];
-->pie(x)
```

执行以上指令，得到如图 5.22 所示的图像窗口。

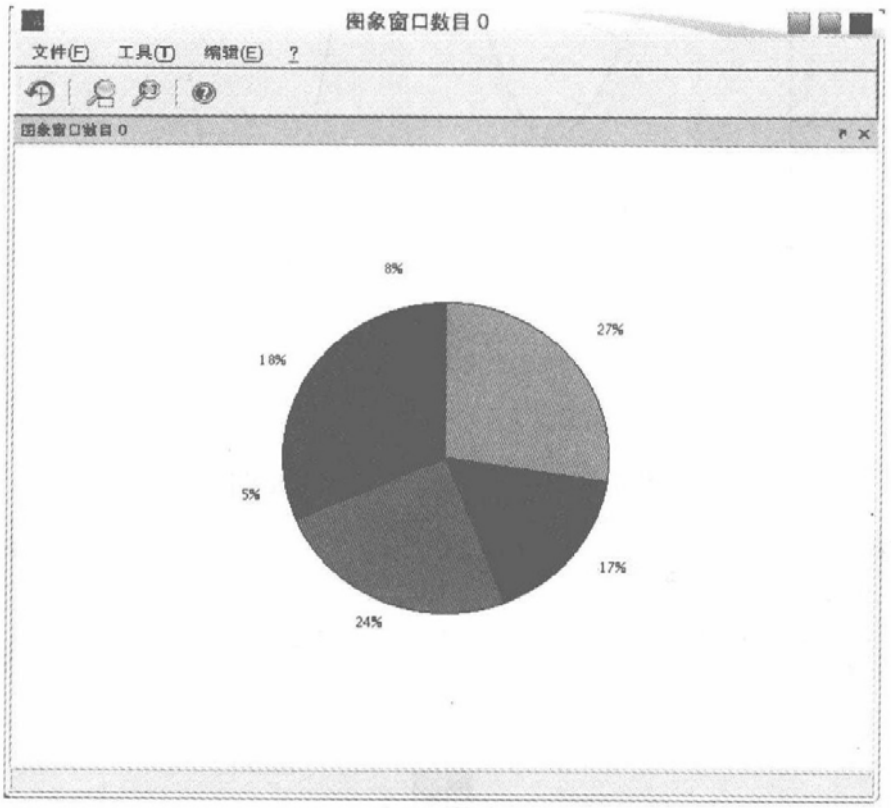


图 5.22 利用 pie 指令绘制饼图

5.3 三维图形的绘制

5.3.1 三维曲面的绘制

Scilab 绘制三维曲面的基本指令是 plot3d。与 plot 指令一样，它也是一个内部函数。在 Scilab 中，输入指令 plot3d()可以观看该函数的演示。下面介绍用 plot3d 指令绘制三维图形时的基本格式。

1. plot3d(x,y,z[,theta,alpha,leg,flag,ebox])

用该指令绘制 z=f(x,y)的曲面。

其中：x,y 表示 x 轴和 y 轴的行向量，必须是单调递增的。如果 x 是长度为 n1 的行向量，y 是长度为 n2 的行向量，则 z 是 n1×n2 的矩阵，z(i,j)是点(x(i),y(j))上的曲面值。

参数 theta 和 alpha 是观察点(视点)的球坐标，是以角度为单位的实数值。

参数 leg 定义每个坐标轴的标题，@是轴标题的分隔符。如 X@Y@Z，表示 3 个坐标轴的标题分别是 X，Y 和 Z。

参数 flag 是包含 3 个参数的向量，flag=[mode,type,box]，

其中每个参数的具体意义如下：

mode：图形隐含部分的处理方法。

mode>0　去除曲面的隐含部分，曲面用颜色模式绘制，mode 是所用颜色的索引号，具体见 getcolor 函数；

mode=0　绘制曲面的隐含部分；

mode<0　只用颜色或模式试绘制曲面的阴影，每个 id 的意义参见 xset() 函数。

type：图形缩放比例。

type=0　利用当前缩放比例绘图(由先前调用 param3d，plot3d，contour 或 plot3d1 等函数时设定的)；

type=1　用 extreme aspect ratios 自动调节 3d 包络盒的比，边界由 ebox 的值指定；

type=2　用 extreme aspect ratios 自动调节 3d 包络盒的比，边界用给定的数据计算；

type=3　3d 包络盒由 ebox 参数设定，与 type=1 类似；

type=4　3d 包络盒由给定的数据设定，与 type=2 类似；

type=5　3d 扩展包络盒由 ebox 参数设定，与 type=1 类似；

type=6　3d 扩展包络盒由给定的数据设定，与 type=0 类似。

box：三维图形周围的包络盒参数。

box=0　围绕图形的包络盒不画；

box=1　与 box=0 同；

box=2　仅画曲面后面的轴；

box=3　绘制包围曲面的包络盒，添加每个轴的标题；

box=4　绘制包围曲面的包络盒，添加轴和标题。

参数 ebox：当 flag 中的 type 标记为 1 时使用。用向量[xmin,xmax,ymin,ymax,zmin,zmax]指定绘制图形的边界。

2. plot3d(x,y,z,<opt_args>)

这种调用格式与第一种类似，参数<opt_args>表示一系列的声明 keyl=valuel,key2=value2,…，这里 keyl,key2,…，可能是上面的参数 theta,alpha,leg,flag,ebox 中之一。可选择的参数 theta,alpha,leg,flag,ebox 用一个声明序列 keyl=valuel,key2 =value2 的方式来定义，这样就不需要特别遵守在上一种调用方式中规定的各个参数的顺序了。

例 5.15 绘制 f(x,y)=sin(x) * cos(y)在[0,2π] ×[0,2π]上的图像。

```
-->t=[0:0.3:2*%pi]';
-->z=sin(t)*cos(t');
-->plot3d(t,t,z,35,45,"X@Y@Z",[2,2,4])
```

执行以上指令，得到如图 5.23 所示的图像窗口。

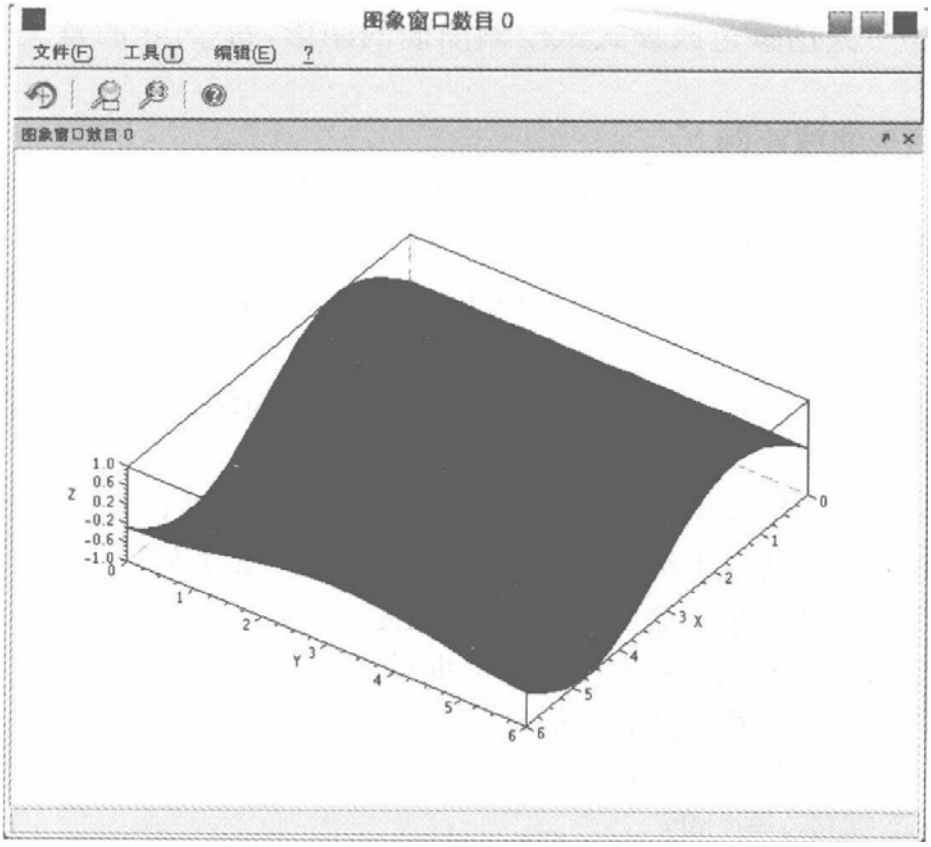


图 5.23 利用 plot 3d 指令绘图

例 5.16 绘制 z(x,y)=x^3+y^3 在[−10,10] ×[−10,10]上的图像。

```
x=-10:0.1:10;
y=x;
for i=1:201
  for j=1:201
    z(i,j)=x(i)^3+y(j)^3;
```

```
    end
  end
  plot3d(x,y,z,35,5,"X@Y@Z",[2,2,4])
```

执行以上指令,得到如图 5.24 所示的图像窗口。

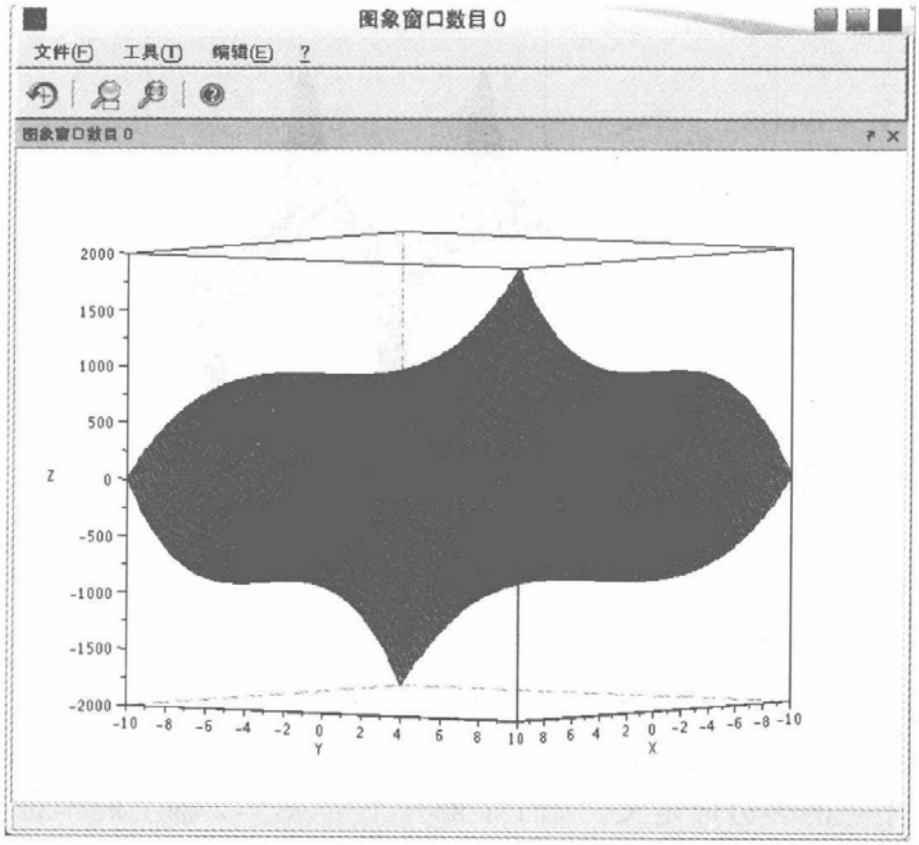


图 5.24 利用 plot3d 指令绘图

例 5.17 绘制 z(x,y)= sin(x)^3 * (−2^y)在[0,4π] ×[0,4π]上的图像。

```
x = 0 : %pi/30 : 4 * %pi;
y = x;
z = zeros(121,121);
for i = 1 : 121
  for j = 1 : 121
    z(i,j) = (sin(x(i)))^3 * ( - 2^y(j));
  end
end
plot3d(x,y,z,60,5,"X@Y@Z",[2,2,4])
```

执行以上指令,得到如图 5.25 所示的图形窗口。

3. plot3d(xf,yf,zf[,theta,alpha,leg,flag,ebox])

用这种调用方式绘制由一组面片组成的曲面。通过替换由[xf1 xf2 …],[yf1

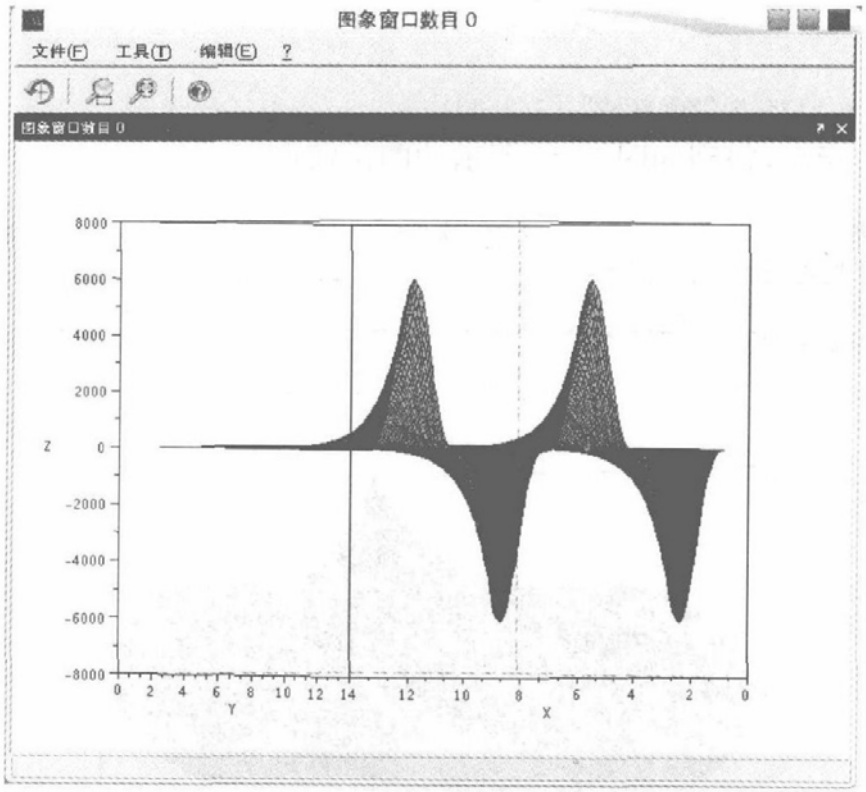


图 5.25 利用 plot3d 指令绘图

yf2 …]和[zf1 zf2 …]组成的矩阵,能绘制多个图。参数 xf,yf 和 zf 分别表示大小为 nf×n 的矩阵,用于定义组成曲面的小面片。共包含 n 个小面片,每个面片 i 用一个具有 nf 个点的多边形定义。第 i 个面片上的 x,y,z 轴上的坐标由 xf(:,i),yf(:,i)and zf(:,i)分别给定。其余参数与上面相同。

在使用这种调用格式的时候,可以用函数 genfac3d 来生成曲面上各小面片上的点的坐标。genfac3d( )的调用格式如下:

[xx,yy,zz]=genfac3d(x,y,z[,mask])

参数说明如下:

xx,yy,zz:4 行(n−1)×(m−1)列的矩阵,xx(:,i),yy(:,i)及 zz(:,i)分别是第 i 个四边形小面片上四点的 x,y,z 方向的坐标。

x:维数为 m 的 x 方向的坐标向量。

y:维数为 n 的 y 方向的坐标向量。

z:m×n 阶的矩阵,z(i,j)为曲面 z=f(x,y)在(x(i),y(j))点的值。

mask:和 z 有相同阶数的布尔型选择矩阵,用来选定由面片来表示 z 的元。

genfac3d:用来计算三维曲面 z=f(x,y)的四边形小面片的坐标。

4. plot3d(xf,yf,zf,<opt_args>)

这种调用方式的参数分别与上面调用方式中的参数意义相同,仅仅是将 the-

ta,alpha,leg,flag,ebox 以参数格式 opt_args 表示出来。

例 5.18 利用 genfac3d()生成 z=sinx * cosy 的小曲面片坐标,绘出该曲面的图像。

```
-->t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
-->[xx,yy,zz]=genfac3d(t,t,z);
-->plot3d(xx,yy,zz)
```

执行以上指令,得到如图 5.26 所示的图像窗口。

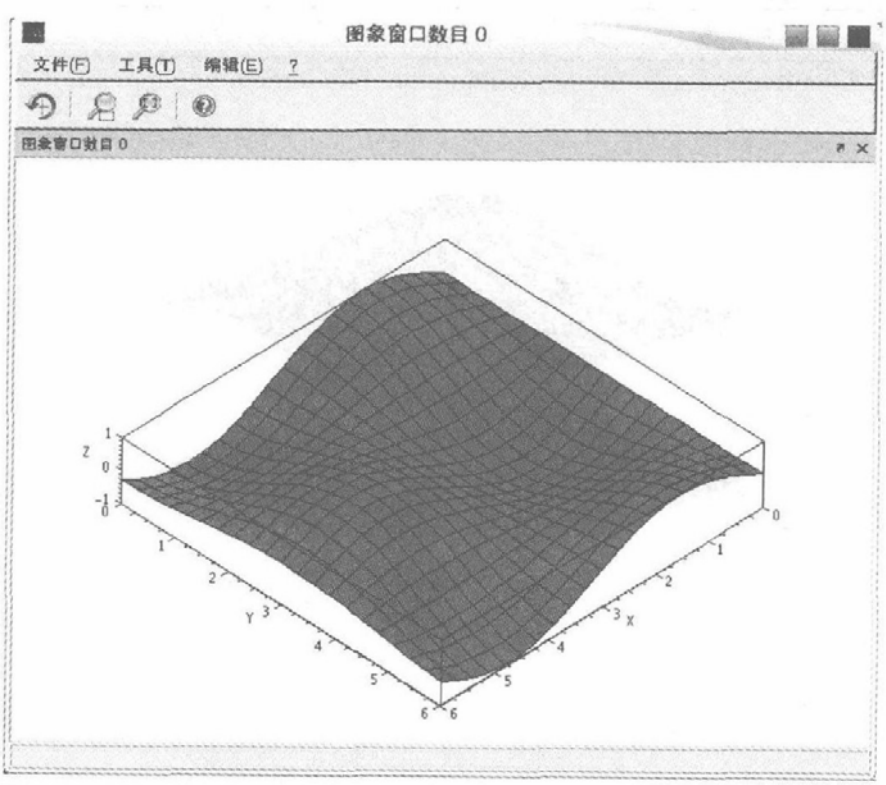


图 5.26 利用 plot 3d 指令绘制 2=sin x * cosy

5. plot3d(xf,yf,list(zf,colors)[,theta,alpha,leg,flag,ebox])

利用 list(zf,colors),而不是 zf,能给每个面片设定一特定的颜色。这里,colors是 n 维向量。如果 colors(i)是正的,则它给出面片 i 的颜色,面片边缘用当前的线型和颜色绘制;如果 colors(i)是负的,则使用颜色索引绝对值。应用 xset()方式可以找到对应各种颜色的索引值。

面片的颜色也可以通过内插得到。在这种情况下,colors 必须是一个大小为 nf×n 的矩阵,给定每个面片边界的颜色。面片的颜色由边缘颜色内插得到。

6. plot3d(xf,yf,list(zf,colors),<opt_args>)

这种调用方式的参数分别与上面调用方式中的参数意义相同。

例 5.19 利用 plot3d( )在同一坐标系中绘制两张曲面的图像。

```
-->x = 0 : 0.1 : 2 * %pi;
-->y = x;
-->z = sin(x)' * cos(y);
-->[xx,yy,zz] = genfac3d(x,y,z);
-->plot3d([xx xx],[yy yy],list([zz zz + 9],[4 * ones(1,3844),5 * ones(1,3844)]))
```

执行以上指令，得到如图 5.27 所显示的图像窗口。

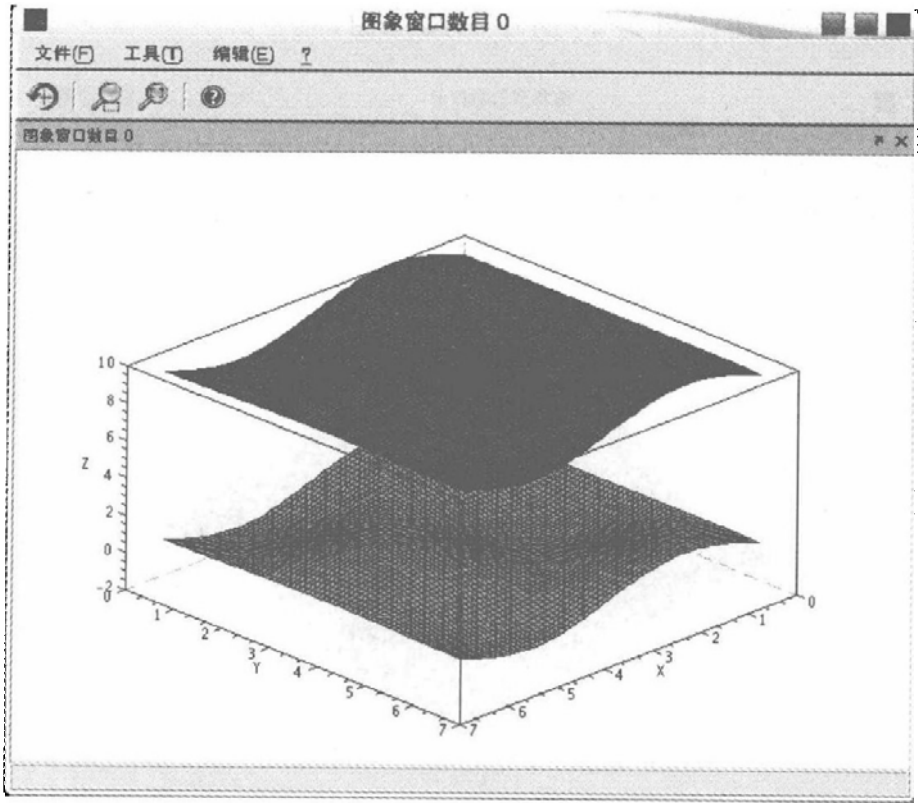


图 5.27 在同一坐标系中绘制两张曲面

5.3.2 三维曲线的绘制

1. param3d 函数

函数 param3d 用于绘制由坐标向量 x,y,z 定义的空间参数曲线。输入指令 param3d()可以观看该函数的演示。下面介绍用 param3d 函数绘制三维图形时的基本调用格式如下：

param3d(x,y,z[theta,alpha,leg,flag,ebox])

其中：

参数 x,y 和 z 是具有相同大小的向量，分别表示确定参数曲线各点的 XYZ 坐标。

参数 theta 和 alpha 是观察点(视点)的球坐标，以角度为单位的实数值。

参数 leg 定义每个坐标轴的标题，@是轴标题的分隔符。例如 X@Y@Z，表示

3 个坐标轴的标题分别是 X,Y 和 Z。

参数 flag 是包含 2 个参数的向量,flag=[type,box],其中每个参数的具体意义如下:

(1) type:整数变量,用于设定图形缩放比例。

type=0 利用当前缩放比例绘图(由先前调用 param3d,plot3d,contour 或 plot3d1 等函数时设定的)。

type=1 用 extreme aspect ratios 自动调节 3d 包络盒的比,边界由 ebox 的值指定。

type=2 用 extreme aspect ratios 自动调节 3d 包络盒的比,边界用给定的数据计算。

type=3 3d 包络盒由 ebox 参数设定,与 type=1 类似。

type=4 3d 包络盒由给定的数据设定,与 type=2 类似。

type=5 3d 扩展包络盒由 ebox 参数设定,与 type=1 类似。

type=6 3d 扩展包络盒由给定的数据设定,与 type=2 类似。

(2) box:三维图形周围的包络盒参数。

box=0 围绕图形的包络盒不画。

box=1 与 box=0 相同。

box=2 仅画曲面后面的轴。

box=3 绘制包围曲面的包络盒,添加每个轴的标题。

box=4 绘制包围曲面的包络盒,添加轴和标题。

参数 ebox 是当 Flag 中的 type 标记为 1 时使用。用向量[xmin,xmax,ymin,ymax,zmin,zmax]指定绘制图形的边界。

例 5.20 利用 param3d1 绘制螺旋线。

```
-->t=0:0.1:5*%pi;
-->param3d(sin(t),cos(t),t/10,35,45,"X@Y@Z",[2,3])
```

执行以上指令,得到如图 5.28 显示的图像窗口。

2. param3d1 函数

函数 param3d1 与函数 param3d 类似,也是一个用于绘制空间参数曲线的函数,其基本调用格式如下:

param3d1(x,y,z,[theta,alpha,leg,flag,ebox])

paraln3d1(x,y,list(z,colors),[theta,alpha,leg,flag,ebox])

其中,x,y,z 为具有相同大小 nl×nc 的矩阵。矩阵的列 i 对应第 i 条曲线。可以用表 list(z,colors)指定每条曲线的颜色,而不是用 z,这里颜色是一个大小为 nc 的向量。如果 colors(i)是负的,则用 id 为 abs(style(i))+1 的标识(mark)绘制图形。

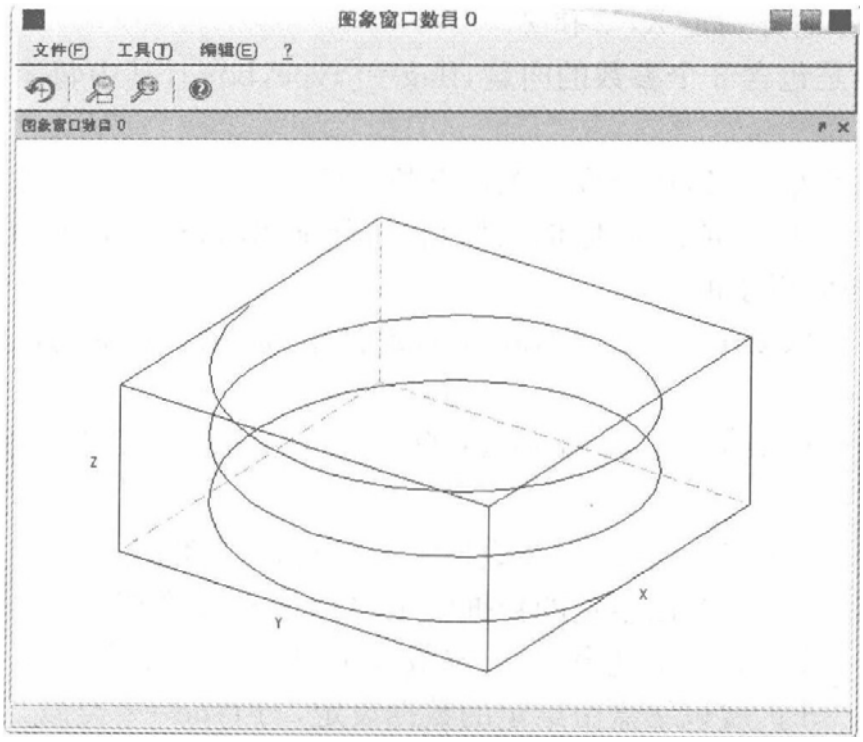


图 5.28 利用 param3d1 绘制螺旋线

例 5.21 利用 param3d1 绘制螺旋线。

```
-->t=[0:0.1:5*%pi]';
-->param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],..
          list([t/10,sin(t)],[3,2]),35,45,"X@Y@Z",[2,3])
```

执行以上指令,得到如图 5.29 所示的图像窗口。

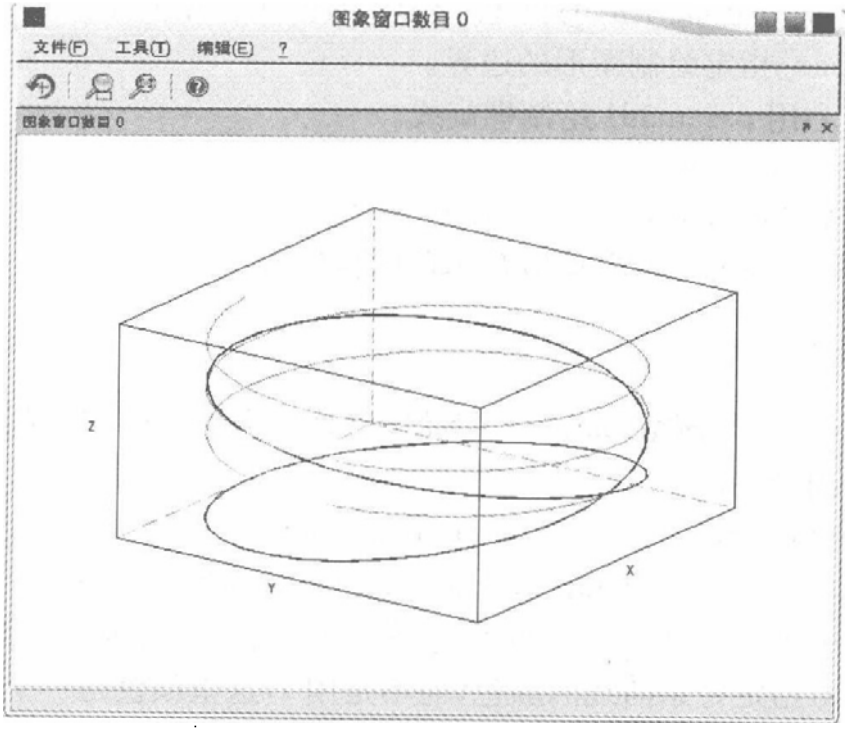


图 5.29 利用 param3d1 绘制螺旋线

5.4 图形修饰处理

前面两节比较详细地介绍了 Scilab 的二维和三维绘图，在绘图的过程中如果需要对图形进行修饰处理，如视点、坐标轴、背景色、线型等进行设置，就需要用到绘图参数与色图的设定。本节介绍绘图全局参数以及色彩和线型的设定，其中有些绘图属性（如视点、坐标轴的显示等）可以通过绘图指令（如 plot、plot2d、plot3d 及 param3d）中的参数来控制，而有些绘图属性（如颜色、线型等）需要通过全局参数来设置。

5.4.1 绘图全局参数的设置

绘图全局参数通过指令 xset( )设定。如果直接在 Scilab 主窗口中输入 xset( )，则会打开一个空白的绘图窗口，用户可以在程序中通过带有参数设置的 xset 指令来对绘图中的全局参数进行设置。

xset( )的调用格式为：

xset(choice-name,x1,x2,x3,x4,x5)

其中 choice-name 为字符串变量，用于设置所设置参数的类型，x1，x2，x3，x4，x5 依赖于 choice-name 设置。

xset("alufunction",number)

用于设置绘图的逻辑功能。使用的逻辑功能是由 X1 设定的。常用的值分别为：3 表示复制（默认），6 表示动画，0 表示清除。详情请参阅 alufunctions。

xset("auto clear","on"|"off")

设置图形的自动清除模式。当自动清除模式设置为“on”，后续绘图将不进行叠加，即在每次执行更高一层的绘图函数之前执行了一次 clf()操作（图形窗口被清除及相关的图形记录被删除）。默认值是“关闭”。

xset("background",color)

Set the background color of the current Axes object. The color argument is the colormap index of the color to use.

设置当前轴对象的背景色。颜色参数是所使用颜色的颜色表的索引值。

xset("clipping",x,y,w,h)

设置裁剪区域（在图形窗口中可以绘图的区域），是一个矩形；x，y 表示最上角点的坐标（即绘图起始点），w 与 h 分别表示绘图的宽度与高度。此函数使用当前的绘图坐标系。

xset("color",value)

设置填充线或文字绘图函数的默认颜色。取值在区间[0,whiteid]中的一整数。0 用于填充黑色，whiteid 用于填充白色。whiteid 的值可以通过 xget("white")获得。

xset("colormap",cmap)

设置颜色表为 m × 3 的矩阵。m 是颜色种数，即有 m 种颜色。色号 i 由一个 3 元组 cmap(i,1)，cmap(i,2)，cmap(i,3)给定，分别对应红色，绿色和蓝色，其取值介于 0 和 1 之间。

xset("dashes",i)

在黑白模式(xset("use color",0))，设置样式风格为 style i(实线为 0)。在彩色模式(xset("use color",1))是用来设置线，标记和文字颜色。当此关键字已过时，请使用 xset("color",i)或 xset(" line style ",i)代替。

xset("default")

将图形参数重置为默认值。

xset("font",fontid,fontsize)

设置当前字体与字体的大小。请注意 fontsize 不仅适用于 fontid，并且适用于所有字体。

xset("font size",fontsize)

设置字体大小。

xset("foreground",color)

设置当前轴对象的前景色。颜色参数是所使用颜色的颜色表的索引值。

xset("fpf",string)

设置功能的数字显示在轮廓浮点格式。字符串是一个字符串给在 C 格式的语法(例如格式字符串="%.3 f"的)。使用字符串=""切换回默认的格式。

xset("hidden3d",colorid)

设置为落后面临 plot3d 面临色号。colorid = 0 抑制落后面临图纸面临三维物体。这在技术上称为 'culling'，加快了封闭表面渲染。

xset("line mode",type)

这个函数是用来设置画线模式。绝对模式设置= 1 型和 2 型糖尿病的相对= 0 模式。(警告：该模式类型= 0 的错误)

xset("line style",value)

当前行设置格式(1：固体，> 1 的虚线)。

xset("mark",markid,marksize)

目前标志设置和当前标记的大小。使用 xset()来查看标记。请注意，marksize 不仅适用于 markid 所有标记。绘图标识类型如图 5.30 所示。

xset("mark size",marksize)

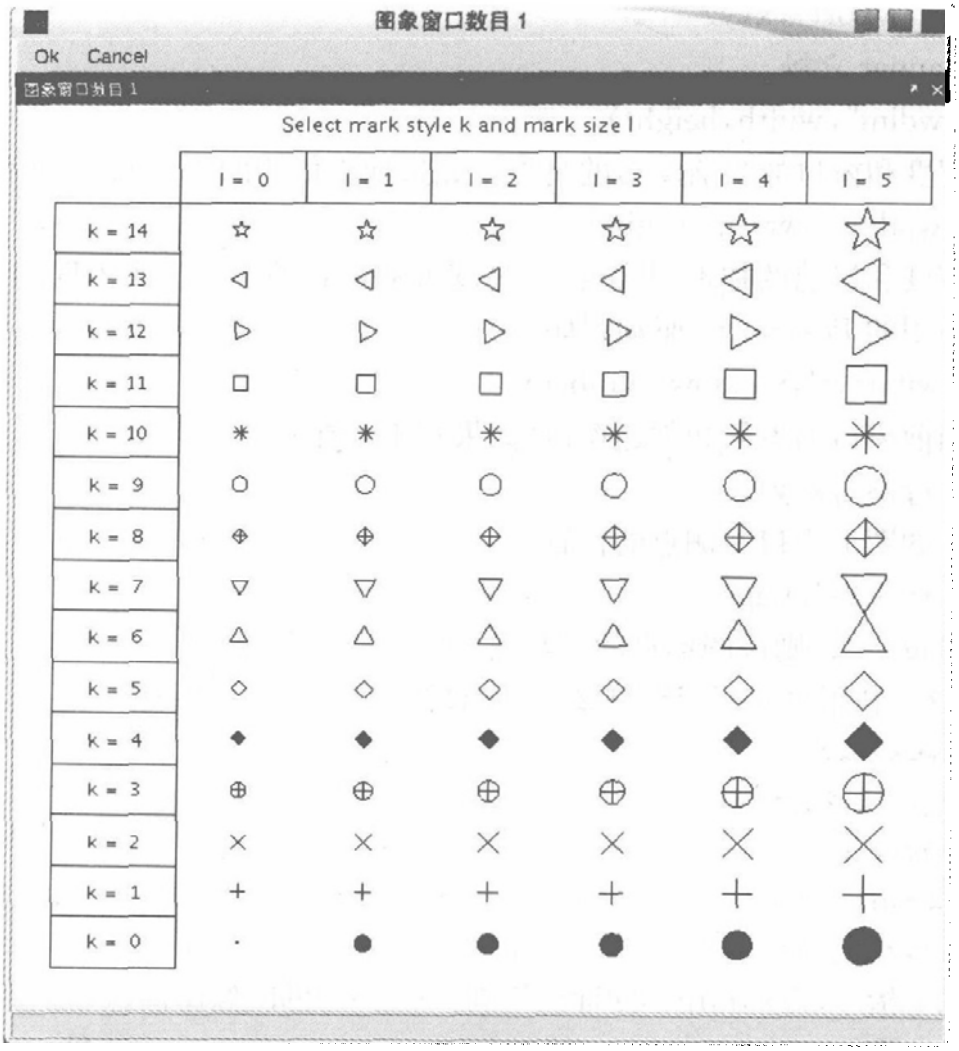


图 5.30 Scilab 绘图标识类型

设置标志的大小。

xset("pattern",value)

填充功能设置为当前模式。值在区间[0,whiteid]预计一整数。0 用于填充黑色和白色 whiteid。该 whiteid 价值可与 xget("白")。"模式"等同于"颜色"。

xset("pixmap",flag)

如果 flag = 0 的图形直接显示在屏幕上。如果 flag = 1 的图形是做一个像素映射和发送到与命令 xset 图形窗口("wshow")。被清除的像素映射用命令 xset("wwpc")。请注意,通常的命令的 clf()也清除像素映射。

xset("thickness",value)

设置在像素线厚度(0 和 1 有相同的含义:1 像素的厚度)。

xset("use color",flag)

如果 flag = 1,则 xset("模式")或 xset("破折号")将用于以更改默认的颜色或图形填充图案。如果 flag = 0,切换回灰色和破折号模式。

xset("viewport",x,y)

设置 panner 立场。

xset("wdim",width,height)

设置宽度和窗口的当前图形的高度。此选项不使用的 PostScript 驱动程序。

xset("wpdim",width,height)

设置宽度和当前的物理图像窗口(高度可从模式的实际大小不同 wresize 1)。此选项不使用的 PostScript 驱动程序。

xset("window",window-number)

设置当前窗口的编号和创建窗口(如果它不存在)。

xset("wpos",x,y)

设置上的图形窗口左侧点的位置。

xset("wresize",flag)

如果 flag = 1,则该图形自动调整大小以填充图像窗口。

例 5.22 利用不同标识绘制 2 * cos(t)在[−π,π]上的图像。

```
xset("mark size",0)
t = - %pi : 0.2 : %pi;
u = 2 * cos(t);
v = [ - 2 + u; - 1.5 + u; - 1 + u; - 0.5 + u;u;0.5 + u;1 + u;1.5 + u;2 + u;]';
plot2d(t,v,style = [ - 9, - 8, - 7, - 6, - 5, - 4, - 3, - 2, - 1])
```

执行以上指令,得到利用不同标识绘制 2 * cos(t)的图像窗口,如图 5.31 所示。

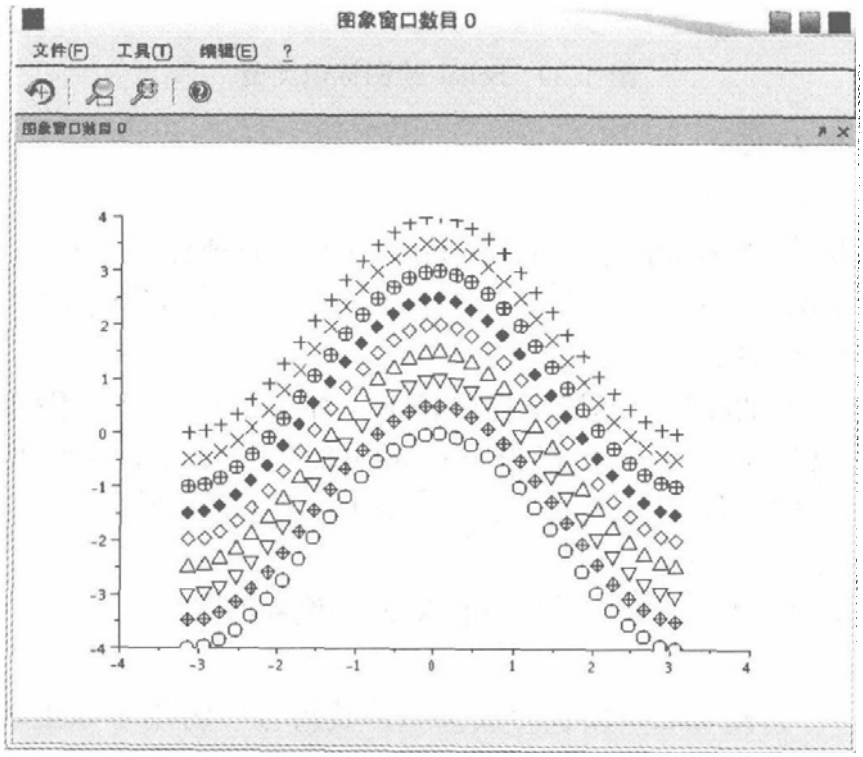


图 5.31 利用不同标识绘图

5.4.2 色图的设定

在绘图时可以通过色图设定来实现多种色彩绘制图形。色图是一个 m×3 的矩阵，其元素在 0 到 1 之间取值。该矩阵的第 i 行表示有 cmap(i,1)，cmap(i,2)，cmap(i,3)所对应的红、绿、蓝三原色混合而得的颜色。默认情况下色图定义了 32 种颜色。可以通过 getcolor()指令得到当前色图设定，并且可以得到色图上各种颜色的索引号。默认情况下色图设定如图 5.32 所示。

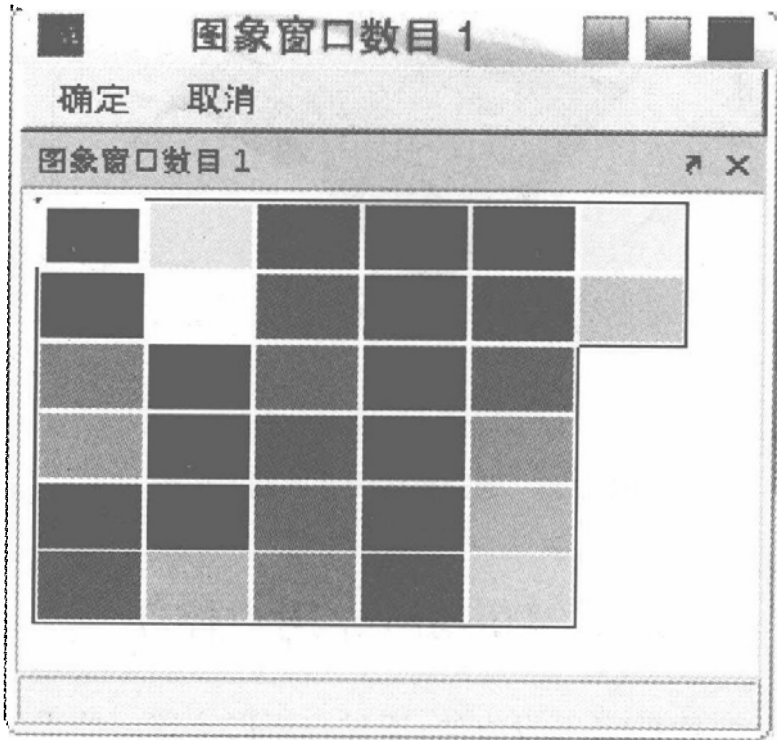


图 5.32 默认情况下色图的设定

例 5.23 利用色图设置对所绘制图形的颜色进行改变。

```
-->m = 228;
-->n = fix(3/8 * m);
-->r = [(1:n)'/n;ones(m - n,1)];
-->g = [zeros(n,1);(1:n)'/n;ones(m - 2 * n,1)];
-->b = [zeros(2 * n,1);(1:m - 2 * n)'/(m - 2 * n)];
-->h = [r g b];
-->xset("colormap",h)
-->plot3d1()
```

执行以上指令后，得到如图 5.33 所示的图像窗口。

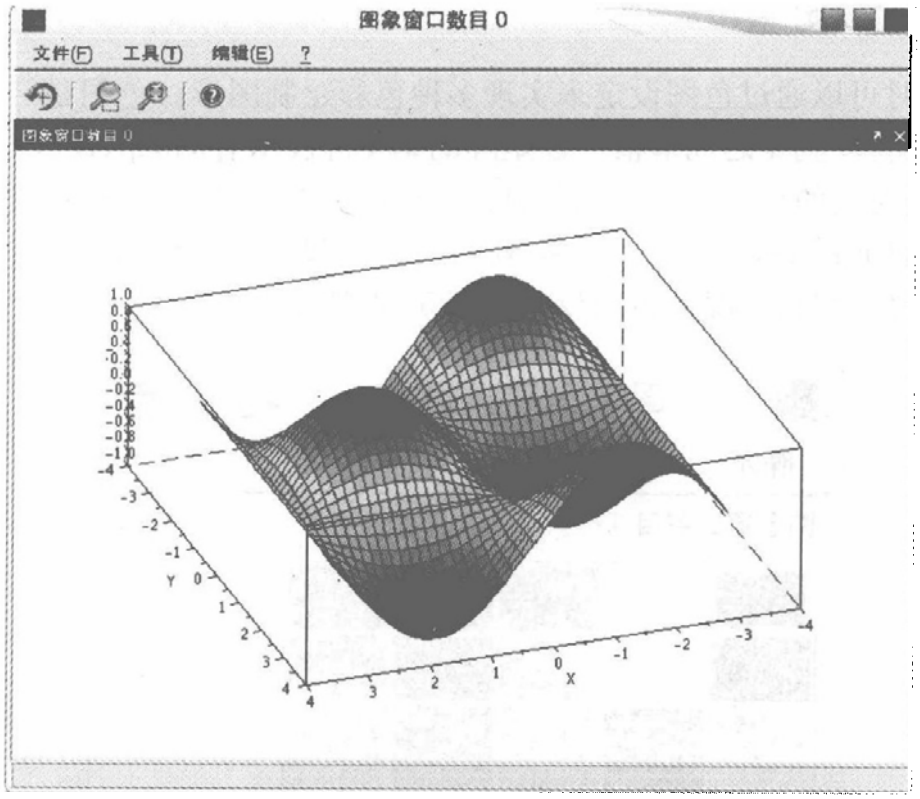


图 5.33 通过色图设置绘制的图形

5.5 Uicontrol 用户控件

本节介绍 Scilab 的 GUI 开发方法，通过它可以实现一些界面化的程序设计，使得 Scilab 的界面编程在教学过程中能有很好地体现。

要做到在一个应用程序中有一个友好的界面，通常的做法是使用图形界面。在使用图形界面时，用户可以不知道应用程序在执行一些什么命令，而只需知道用户图形界面组件的使用方法，也不需要知道应用程序命令怎么样执行，而只要通过与界面的简单交互就可得到所指定行为正确的执行结果。这大大增加了用户使用应用程序的方便性，使学习和使用大为容易。

Scilab 中的图形用户界面(GUI)是一种包含多种对象的图形窗口。用户必须对每一个界面进行布局和编程，从而使用户激活 GUI 每个对象时都可以执行相应的行为。

uicontrol 指令主要用来创建一个用户界面控件对象，这些控件对象可以通过回调(callback)属性来进行回调行数编程，实现相应的用户行为。回调函数的书写是实现 GUI 界面交互功能的真正体现，GUI 的各种功能正是在依靠各个控件的具体回调函数实现的。所以说，对各种控件功能及其属性的了解归根到底都是为了

编写回调函数的需要。

uicontrol()的调用格式如下：

```
h = uicontrol(PropertyName,PropertyValue,...)
h = uicontrol(parent,PropertyName,PropertyValue,...)
h = uicontrol(uich)
```

功能

在绘图中创建一个对象。

如果绘图的句柄(第一个参数)是给定的，图形对象将在绘图中创建。如果没有给定，图形对象创建在当前的绘图中(可以通过调用 gcf()获得)。如果没有当前绘图，则在创建图形对象之前创建一个绘图。

当控件创建后，其具有相应的值作为参数的属性也就设置好了。它相当于创建 uicontrol，然后使用 set()命令设置其属性。在调用 uicontrol()时设置属性通常更有效。特别是针对"Style"属性。事实上，此属性的默认值是"Pushbutton"。如果在创建时没有设置它，将创建一个按钮；当调用 set(h,"Style",...)指令时，它会转化为另一种 uicontrol。Scilab 和所有的图形对象通过属性机制进行交互。因此，要创建合适的 uicontrol，用户必须要知道如何使用属性字段。

h = uicontrol(PropertyName, PropertyValue,...) 创建一个 uicontrol 并分配指定的属性和值给它。如果用户没有指定，所以属性将被设置为默认值。默认 uicontrol 的格式是"Pushbutton"。默认的父对象是当前的绘图。请参阅有关这些属性和其他属性的详细信息部分。

h = uicontrol(parent, PropertyName, PropertyValue,...) 创建一个由句柄、父对象指定的对象 uicontrol。如果你还为指定不同的值，该值具有优先权。父对象是绘图的句柄。

参数

BackgroundColor

[1,3] real vector or string (1×3 实向量或字符串)

uicontrol 对象的背景色。颜色指定为红，绿，蓝值。这些值是介于[0,1]之间的实数。颜色可以给定为一个实向量，即[R,G,B]或是一个字符串，其中每个值之间用"|"分开，即"R|G|B"。

Callback

String(字符串)

这个属性声明了当用户触发 uicontrol 对象(如点击按钮)的时候 Scilab 解释器所执行的字符串。

Enable

{on} | off

启用或禁用 uicontrol。如果此属性设置为"on"(默认),uicontrol 可以运行,但如果此属性设置为"关闭"时,uicontrol 不会响应鼠标操作,将变成灰色。

FontAngle

{normal} | italic | oblique

一个包含一些文本的控件,该属性用于设置字体的倾斜。

FontSize

Scalar

一个包含一些文本的控件,该属性用于设置在 FontUnits 中的字体大小。

FontUnits

{points} | pixels | normalized

一个包含一些文本的控件,该属性用于设置被指定字号的单位。

FontWeight

light | {normal} | demi | bold

一个包含一些文本的控件,该属性设置所使用的字体粗细。

FontName

String

用于选择显示选定控件的文本的字体名称。

ForegroundColor

[1,3] real vector or string

uicontrol 对象的前景色。颜色指定为红,绿,蓝值。这些值是介于[0,1]之间的实数。颜色可以给定为一个实向量,即[R,G,B]或是一个字符串,其中每个值之间用"|"分开,即"R|G|B"。

HorizontAlalignment

left | {center} | right

设置在 uicontrol 中文本的水平对齐方式。此属性仅对文本、编辑和复选框起作用。

ListboxTop

Scalar

对于一个 ListBox，此属性告诉哪个列表项在列表的可见区域的第一行显示。

Max

Scalar

指定的可以设置的最大值“Value”属性。但是它在每个 uicontrol 中有不同的含义：

- 复选框：最大值是当控件被选中时“Value”属性的值。
- 滑块：滑块的最大值。
- 列表框：如果(最大值－最小值)＞ 1，列表允许多重选择，否则不是。

Min

Scalar

指定的可以设置的最小值“Value”属性。但是它在每个 uicontrol 中有不同的含义：

- 复选框：最小值是当控件没有被选中时“Value”属性的值。
- 滑块：滑块的最小值。
- 列表框：如果(最大值－最小值)＞ 1，列表允许多重选择，否则不是。

Parent

Handle

unicontrol 父对象的句柄。更改此属性允许将控件从一个绘图移动到另一个绘图。

Path

此属性是不再被支持。

Position

[1,4] real vector or string

此属性用于设置或获取控件的几何配置。它是一个向量[x y w h]，其中各字母分别表示左底角的 x 坐标、左底角的 y 坐标、uicontrol 的宽度和高度；或者是一个字符串，其中每个值由" | "分隔开，即"x|y|w|h"。这些元素由“Units”属性确定。

宽度和高度值确定滑块的方向。如果宽度比高度大，则滑块导向的水平，否则滑块是垂直方向。

Relief

flat | groove | raised | ridge | solid | sunken

uicontrol 边界的外观：

- 按钮：“Relief”属性的默认值为“raised (凸出)”。
- 编辑：“Relief”属性的默认值为“sunken (下沉)”。

· 其他格式："Relief"属性的默认值为"flat（平的）"。

SliderStep

[1,2] real vector

[small big]，小步长表示点击滑块槽或轻击键盘上的方向箭头时（当滑块被聚焦时）的滑块移动的步长，大步长是使用 Ctrl-键盘方向箭头时滑块移动的步长。如果省略了大步，它的默认值是滑操长度的 1/10。

String

String

此属性表示在 uicontrol 中出现的文本（Frame 和 Slider 格式除外）。对于 ListBoxes 和 PopupMenus，取值可以是由"|"分隔的字符串向量或一个字符串。对于文本 uicontrols，这个字符串可以包含 HTML 代码来编排文本。

Style

{pushbutton} | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu

uicontrol 的格式。下面是每一个的简短说明：

- Pushbutton（按钮）：一个长方形的按钮通常用来运行一个回调。
- Radiobutton（单选按钮）：一个按钮标识两个状态。单选按钮是互相排斥的（您的代码必须实现相互排斥的行为）。
- Checkbox（复选框）：一个按钮标识两个状态（用于多个独立的选择）。
- Edit（编辑）：可编辑字符串的区域。
- Text（文本）：一个文本控件（通常是静态的）。
- Slider（滑块）：一个尺度控件，可以用鼠标来设置滚动条在给定范围内的值。
- Frame（框）：一个控件，表示一个可以用来分组相关的控件的区域。
- Listbox（列表框）：一个控件，表示了一个可以滚动的项目清单。该项目可以用鼠标选择。
- Popupmenu：一个按钮，单击时菜单出现。

Tag

String

此属性通常用于识别控件。它允许给它一个"name（名字）"。主要与 findobj()结合使用。

Units

{points} | pixels | normalized

用于设置单位来指定"Position（位置）"属性。

Userdata

Scilab data

可以用于将一些 Scilab 对象(字符串,字符串矩阵,m×n 矩阵)关联到一个 uicontrol。

Value

Scalar or vector

uicontrol 的值。确切含义取决于 uicontrol 的格式:

- CheckBoxes(复选框),Radio buttons(单选按钮):选中时(on)被设置成最大(见上文),未选中时(off)被设置成最小。
- ListBoxes(列表框),PopupMenus:其值是一个索引向量对应于列表中选定项的索引。1 是列表的第一个项目。
- 滑块:由滑块标示的值。

Verticalalignment

top | {middle} | bottom

设置 uicontrol 中文本的垂直对齐。此属性仅对文本和复选框格式起作用。

Visible

{on} | off

设置 uicontrol 是否可见。如果此属性设置为"on"(默认),uicontrol 是可见的,但如果此属性设置为"off"时,uicontrol 在其父绘图中将不会出现。

以上介绍了 Scilab 的用户界面控件对象 unicontrol,下面我们结合它给出一个例子来详细说明控件对象的用法。

结合高中数学中定积分的概念,利用 Scilab 的界面编程来实现利用分割的方法来逼近求曲边梯形面积的直观演示。本例中要求所给的函数 f(x)在区间[a,b]上是单调非负的,通过 unicontrol 中的文本框、编辑框、按钮以及滑条来实现。其中编辑框用来输入端点 a、b 的值以及函数表达式 f(x),其中 f(x)的输入变量为 x,文本框用来显示分别从曲线的上下方逼近曲边梯形面积的值以及区间[a,b]的等分数 n,滑块用于改变区间[a,b]的等分数 n。详细代码如下:

1. 设置绘图

```
function curseur_aire()
global lock slider valueDisplay minDisplay maxDisplay line aedit bedit fcedit;
global ff fig;
```

```
global sliderMin sliderMax sliderSteps;

try
   close(fig)
end
try
  delete(ff);
end
```

2. 建立绘图

```
figWidth = 620; // 图像宽度
figHeight = 800; // 图像高度
fig = figure("Position",[10,10,figWidth,figHeight]);
set(fig,"figure_name","面积计算求积分近似值");
fig.auto_resize = "off";

ff = scf(fig);
ff.background = -2;
axes = gca();
axes.box = "on";
axes.visible = "on";
axes.axes_bounds = [-0.05 -0.05 1.1 0.8];
axes.margins = [0.1 0.1 0.1 0.1];

// 框

frX = 80;  // 框的横坐标
frWidth = 300;   // 框的宽度
uicontrol(fig,"Style","frame",..
                  "Position",[frX,90,frWidth,100],..
                  "BackgroundColor",[0.9,0.9,0.9]);

// 函数

fctWidth = 100;
uicontrol(fig,"Style","text",..
                  "Position" ,[frX + 20,160,fctWidth,20],..
```

```
                    "BackgroundColor",[0.9,0.9,0.9],..
                    "Horizontalalignment","center",..
                    "String","Function :","FontSize",14,"FontWeight","bold");
fcWidth = 150;
fcedit = uicontrol(fig,"Style","edit",..
                        "Position",[frX + 110,160,fcWidth,20],..
                        "String","x^2","FontSize",14,"FontWeight","bold");

// 端点 a

atWidth = 10;  // 端点 a 标签宽度
uicontrol(fig,"Style","text",..
                    "Position" ,[(figWidth-atWidth)/2-190,130,atWidth,20],..
                    "BackgroundColor",[0.9,0.9,0.9],..
                    "Horizontalalignment","center",..
                    "String","a :","FontSize",14,"FontWeight","bold");
aWidth = 40;                        // 端点 a 文本框宽度
aedit = uicontrol(fig,"Style","edit",..
                        "Position",[(figWidth - aWidth)/2 - 150,130,aWidth,20],..
                        "String","0","FontSize",14,"FontWeight","bold");

// 端点 b

btWidth = 10;  // 端点 b 标签宽度
uicontrol(fig,"Style","text",..
                    "Position" ,[(figWidth - btWidth)/2 - 90,130,btWidth,20],..
                    "BackgroundColor",[0.9,0.9,0.9],..
                    "Horizontalalignment","center",..
                    "String","b :","FontSize",14,"FontWeight","bold");
bWidth = 40;                        // 端点 b 标签宽度
bedit = uicontrol(fig,"Style","edit",..
                        "Position",[(figWidth - bWidth)/2 - 40,130,bWidth,20],..
                        "String","5","FontSize",14,"FontWeight","bold");

// 按钮 ok

npokWidth = 40;
uicontrol(fig,"Style","pushbutton",..
```

```
                "Position",[(figWidth-npokWidth)/2-80,100,npokWidth,20],..
                "String","Ok","FontSize",14,"FontWeight","bold",..
                "Callback","newdata_aire");
```

3. 滑条

```
sliderMin=1; //滑条最小值
sliderMax=600; //滑条最大值
sliderX=50;  // 滑条框的横坐标
sliderY=50;  // 滑条框的纵坐标
sliderWidth=500; // 滑条框的长度
sliderHeight=20; // 滑条框的高度
sliderSteps=sliderMax-sliderMin; // 滑条移动的步数
//sliderSteps=100; // 滑条移动的步数

// 建立滑条
slider=uicontrol(fig,"Style","slider",..
                        "Position",[sliderX,sliderY,sliderWidth,sliderHeight],..
                        "Min",0,"Max",sliderSteps,..
                        "callback","update_aire");

// 显示端点最小值
lowerBoundPos =[sliderX-30,sliderY,20,sliderHeight];
uicontrol(fig,"Style","text",..
                "Horizontalalignment","right",..
                "Position",lowerBoundPos,..
              "BackgroundColor", [1 1 1],..
                 "String",string(sliderMin),"FontSize",14,"FontWeight","bold");

// 显示端点最大值
upperBoundPos=[sliderX+sliderWidth+20,sliderY,30,sliderHeight];
uicontrol(fig,"style","text",..
                "Horizontalalignment","left",..
                "Position",upperBoundPos,..
            "BackgroundColor", [1 1 1],..
                 "String",string(sliderMax),"FontSize",14,"FontWeight","bold");

// 显示当前值
valueDisplayPos=[sliderX+sliderWidth/2-50,sliderY-30,150,20] ;
```

```
valueDisplay = uicontrol(fig,"style","text","Horizontalalignment","left",..
      "BackgroundColor", [1 1 1],..
      "Position",valueDisplayPos,"FontSize",14,"FontWeight","bold");

// 显示值
titlePos = [frX + frWidth + 20,160,200,20];
maxDisplay = uicontrol(fig,"style","text","Horizontalalignment","left",..
     "BackgroundColor", [1 1 1],..
     "Position",titlePos,"FontSize",14,"FontWeight","bold");

titlePos = [frX + frWidth + 20,120,200,20];
minDisplay = uicontrol(fig,"style","text","Horizontalalignment","left",..
     "BackgroundColor", [1 1 1],..
     "Position",titlePos,"FontSize",14,"FontWeight","bold");

lock = %t;

endfunction
```

4. 当单击 ok 按钮时激活函数

```
function newdata_aire()
global a b N func hrmin hrmax croit ferreur;
global lock line aedit bedit fcedit;
global ff;

ferreur = %F;
fc = get(fcedit,"String");
try
   deff("[r] = func(x)","r = " + fc);
catch
   messagebox("函数定义时的语法错误!","Error","error","modal");
   ferreur = %T; return;
end

a = get(aedit,"String");
[a,ierr] = evstr(a);
if ierr<>0 then
   messagebox("端点 a 必须是数字","Error","error","modal");
```

```
    ferreur = %T; return;
end
if type(a)<>1|size(a,"*")<>1 then
    messagebox("端点 a 必须是数字","Error","error","modal");
    ferreur = %T; return;
end

b = get(bedit,"String");
[b,ierr] = evstr(b);
if ierr<>0 then
    messagebox("端点 b 必须是数字","Error","error","modal");
    ferreur = %T; return;
end
if type(b)<>1|size(b,"*")<>1 then
    messagebox("端点 b 必须是数字","Error","error","modal");
    ferreur = %T; return;
end

if a >= b then
    ferreur = %T; messagebox("b 必须比 a 大!","Error","error","modal"); return;
end;

//测试函数:考虑非向量语法
n = 1000;
xx = linspace(a,b,n);
try
    for k = 1 : n yy(k) = func(xx(k)); end;
    if ~isreal(yy) then
        messagebox("函数计算过程中产生复数!","Error","error","modal");
        ferreur = %T; return
    end
catch
    [str,ierr] = lasterror();
    select ierr
    case 27 then
      messagebox("函数计算过程中出现分母为 0 的情况","Error","error","modal");
    case 32 then
      messagebox("在计算 ln 或者 tan 的过程中出现奇异值","Error","error","modal");
```

```
    else
      messagebox("函数计算过程中出现错误","Error","error","modal");
    end
    ferreur = %T; return
end
if min(yy) < 0 then
    messagebox("函数必须要大于 0","Error","error","modal");
    fError = %T; return;
end;
croit = %F; decroit = %F;
for k = 2 : n
if yy(k) >= yy(k-1) then croit = %T; else decroit = %T; end;
end
if croit & decroit then
    messagebox("函数必须是单调的","Error","error","modal");
    ferreur = %T; return;
end;

//绘图
drawlater();
scf(ff);
axes = gca();
if ~isempty(axes.children) then
    delete(axes.children);
end
// 当去处 Inf 与 Nan 后的新边界
infxx = find(isinf(xx) == %T); xx(infxx) = []; yy(infxx) = [];
nanxx = find(isnan(xx) == %T); xx(nanxx) = []; yy(nanxx) = [];
infyy = find(isinf(yy) == %T); xx(infyy) = []; yy(infyy) = [];
nanyy = find(isnan(yy) == %T); xx(nanyy) = []; yy(nanyy) = [];
axes.data_bounds = [min(xx),min(yy);max(xx),max(yy)];
plot(xx,yy);
drawnow();
lock = %f; hrmin = [];
update_aire();
endfunction
```

5. 当滑条移动时激活函数

```
function update ()
```

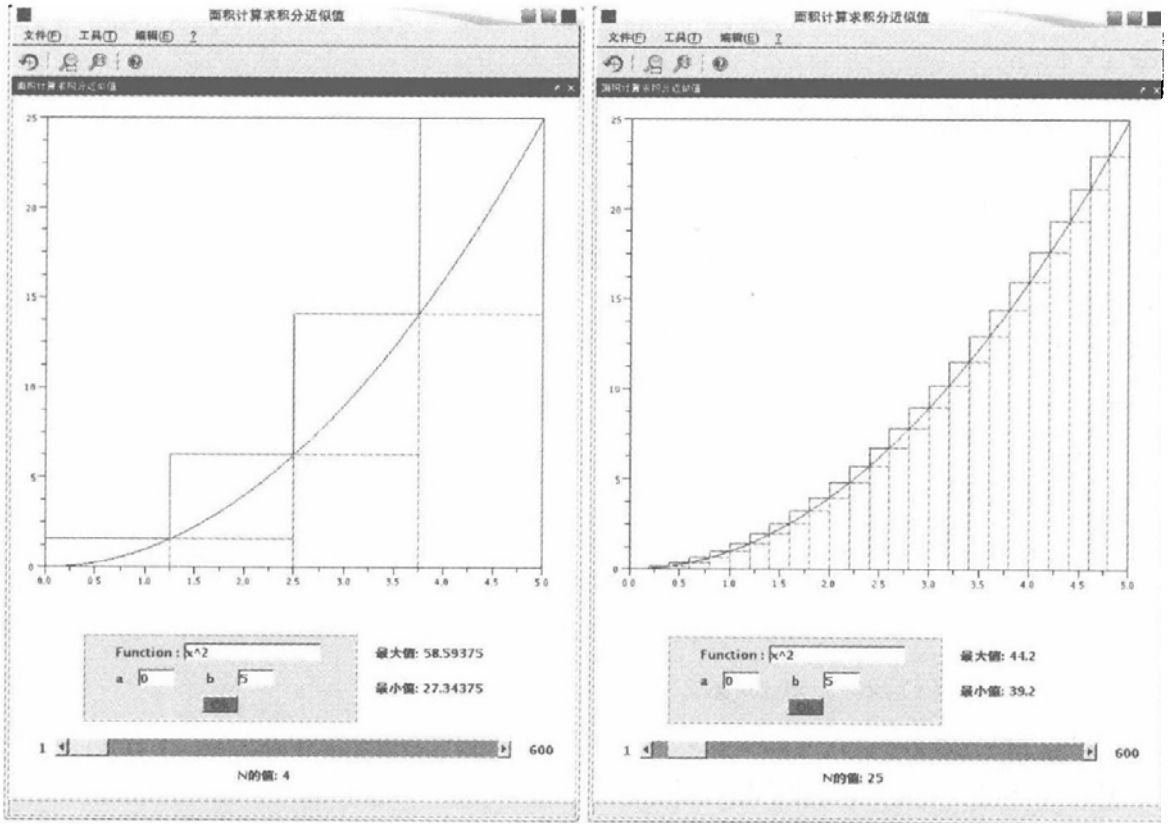
```
global a b N hrmin hrmax croit ferreur;
global lock slider valueDisplay minDisplay maxDisplay line hr;
global sliderMin sliderMax sliderSteps;

// 当加锁时返回
if lock then return end;

// 当输入的数据存在错误时返回
if ferreur then return end;

// 加锁
lock = %t ;
drawlater();
if hrmin <> [] then
   delete(hrmin);
   delete(hrmax);
end

// 显示当前 N 的值
curVal = get(slider,"Value");
N = curVal * (sliderMax-sliderMin)/sliderSteps + 1;
N = floor(N);
set(valueDisplay,"String","Divisions : " + string(N));

// delimitation of the right and display the error
[s,S] = approche(a,b,N);
if croit then   //输入函数递增
   set(maxDisplay,"String","Maximum value: " + string(S));
   set(minDisplay,"String","Minimum value : " + string(s));
else                //输入函数递减
   set(maxDisplay,"String","Maximum value: " + string(s));
   set(minDisplay,"String","Minimum value : " + string(S));
end
drawnow();

// 解锁
lock = %f;
endfunction
```

```
function [s,S] = approche(a,b,n)
global func hrmin hrmax croit;
s = 0;
for k = 0 : n - 1;
  s = s + func(a + k * (b - a)/n)
end
s = s * (b - a)/n;
S = 0;
for k = 1 : n
  x(k) = a + (k - 1) * (b - a)/n; y(k) = func(x(k));
  S = S + func(a + k * (b - a)/n)
end
x(n + 1) = b; y(n + 1) = func(x(n + 1));
S = S * (b - a)/n;
Xmin = [];Ymin = []; Xmax = []; Ymax = [];
if croit then
   for i = 1 : n
     Xmin = [Xmin;x(i);x(i);x(i);x(i + 1)];
     Ymin = [Ymin;0;y(i);y(i);y(i)];
     Xmax = [Xmax;x(i);x(i);x(i);x(i + 1)];
     Ymax = [Ymax;y(i);y(i + 1);y(i + 1);y(i + 1)];
   end
   Xmin = [Xmin;x(n + 1);x(n + 1)];
   Ymin = [Ymin;y(n);0];
   Xmax = [Xmax;x(n + 1);x(n + 1)];
   Ymax = [Ymax;y(n + 1);y(n)];
else
   for j = 1 : n
     i = n + 2 - j;
     Xmin = [Xmin;x(i);x(i);x(i);x(i - 1)];
     Ymin = [Ymin;0;y(i);y(i);y(i)];
     Xmax = [Xmax;x(i);x(i);x(i);x(i - 1)];
     Ymax = [Ymax;y(i);y(i - 1);y(i - 1);y(i - 1)];
   end
   Xmin = [Xmin;x(1);x(1)];
   Ymin = [Ymin;y(2);0];
   Xmax = [Xmax;x(1);x(1)];
   Ymax = [Ymax;y(1);y(2)];
```

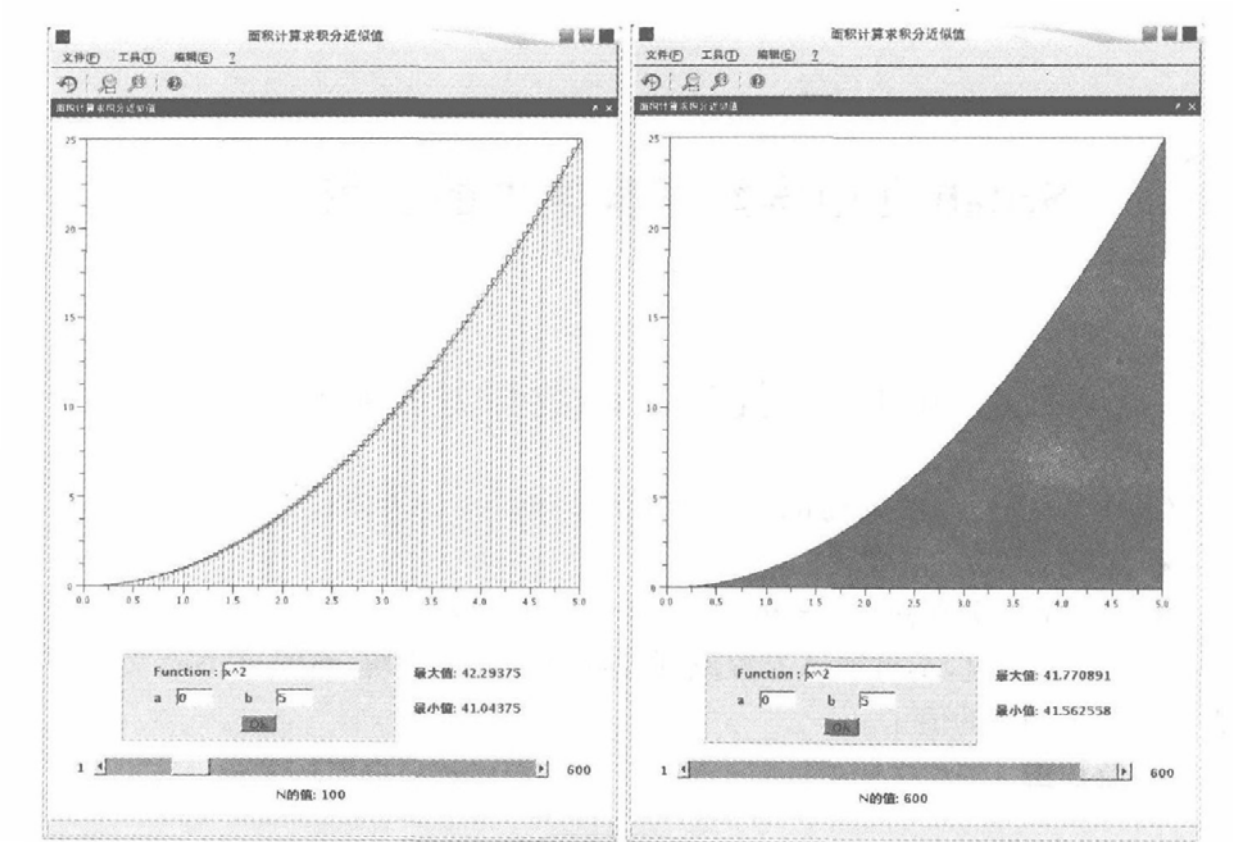
```
end

//绘制折线(矩形)
xpoly(Xmin,Ymin); hrmin = gce();
hrmin. foreground = color("green");
xpoly(Xmax,Ymax); hrmax = gce();
hrmax. foreground = color("red");

endfunction
```

当 a=0,b=5,f(x)=x^2 时的运行结果如图 5.34 所示。

从图中可以看出,随着 N 的值逐渐增大,面积最大值在逐渐减小,而面积最小值在逐渐增大,它们都在逼近曲边梯形的面积 125/3≈41.666。通过图形可以直观地看到随着 N 的增大,所求的面积的变化趋势。



(a) (b)



(c) (d)

图 5.34 图形界面调试效果图

5.6 Scilab 对显卡驱动的要求

Scilab 支持两种显示模式：GLCanvas 与 GLJPanel。

GLJPanel 是默认的显示模式。但是某些操作系统平台、显卡、显卡驱动不支持 GLJPanel 模式，因此只能强制使用 GLCanvas，会出现一个警告。GLCanvas (AWT+OpenGL)是 Java Framework 提供的组件，这种模式将使 Scilab 的某些功能不可用，如"mixing plots with uicontrol"。

可以通过命令 usecanvas 在两种模式中切换，当指定参数为%F 时使用 GLJPanel 模式；指定参数为%T 时使用 GLCanvas 模式。

用户可以根据计算机的显卡品牌，参见 usecanvas 的命令帮助功能加以解决。

6 Scilab 在中学数学教学中的应用

6.1 一元二次函数与方程求根

(1) 求解一元二次方程 $ax^2+bx+c=0(a\neq 0)$ 的根。

◆ **建模**

由于一元二次方程 $ax^2+bx+c=0(a\neq 0)$ 是否有实数解主要是取决于它的 $\Delta=b^2-4ac$ 的值，不过把该方程放到复数范围内来求解，这时就不需要考虑判别式 $\Delta=b^2-4ac$ 的值了。

◆ **流程图**

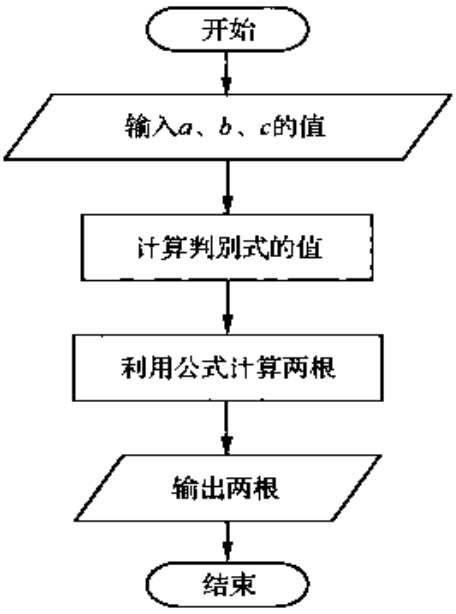


图 6.1 求解一元二次方程流程图

◆ **Scilab 程序**

```
  function  QEOUnknown()          // 定义函数名
clc
clear                             //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';'程序：计算';'描述：';
'计算一元二次方程的根';],['确定']);  // 描述函数的作用及程序介绍
```

```
label1 = ['a :';'b :';'c :';];
B = x_mdialog(['请输入系数 a、b、c 的值 :'],label1,['1';'2';'-3';]);
                                        //预设 a,b,c 的值
a = evstr(B(1));
b = evstr(B(2));
c = evstr(B(3));                        // 提示输入 a,b,c 的值
if B == []
quit
end                                     //若选择取消按钮,则结束

D = messagebox(['确认计算一元二次方程的根?'],['退出','计算']);  //询问是否进行计算
  r = sqrt(b*b-4*a*c);                  //计算判别式
  x1 = (-b+r)/(2*a);                    //计算较小的根
  x2 = (-b-r)/(2*a);                    // 计算较大的根
  disp(x2,"x2 = ",x1,"x1 = ");          //输出两根

d = string(x1);
e = string(x2);                         //将两根符号化

E = messagebox(['该方程的根 x1 、x2 分别为:';d;e;],['完成'])   //输出两根的值
endfunction
```

◆ 程序运行结果

执行此程序,并输入 $a=1,b=2,c=-3$ 得出如图 6.2 所示的结果,a,b,c 的值不同得到的结果就不一样。

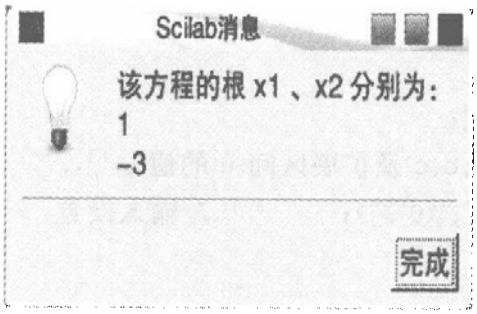


图 6.2 例(1)的执行结果

(2) 计算一元二次函数 $y=ax^2+bx+c(a\neq 0)$ 的对称轴、最值及两根并作出图像。

◆ 建模

一元二次函数 $y=ax^2+bx+c(a\neq 0)$ 的系数一般是实数,不过这里可以把它扩充到复数范围内,所以在求函数图像与 x 轴的交点的横坐标时(及求方程

$ax^2+bx+c=0(a\neq 0)$ 的根),就不用考虑判别式 $\Delta=b^2-4ac$ 的值了。

◆ **流程图**

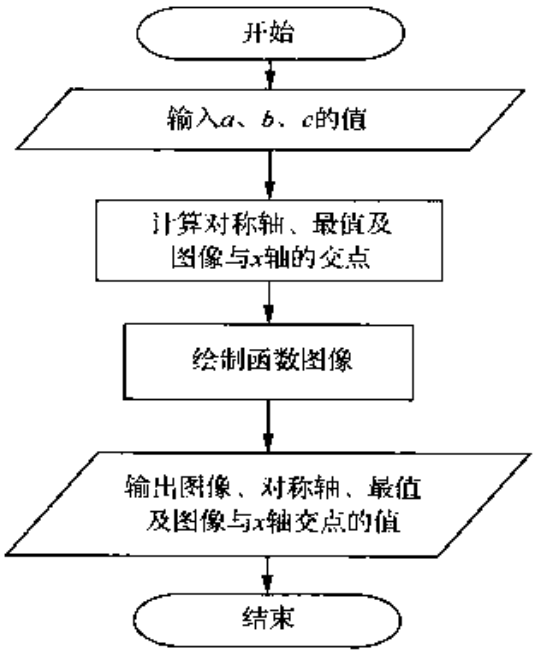


图 6.3 例(2)的流程图

◆ **Scilab 程序**

```
  function  yi_yuan_er_ci_han_shu()  //定义函数名
clc
clear                                  //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';'程序:绘制图形';
'描述:';'计算一元二次函数对称轴、最值及两根并作图像';''],
['确定']);          //介绍程序的作用

label1 = ['a :';'b :';'c :';'u';];
B = x_mdialog(['请输入系数 a,b,c 及扩展区间 u 的值 :'],...
            label1,['1';'0';'0';'10';]);          //输入设置

a = evstr(B(1));
b = evstr(B(2));
c = evstr(B(3));
u = evstr(B(4));
if B = = []
quit
end                        //若选择取消按钮
D = messagebox(['确认计算一元二次函数对称轴、最值及两根?'];],
```

```
['退出','确认']);          //询问是否进行计算

//编程
x0 = -(b/(2*a));                    //计算对称轴的值
y0 = a*x0^2 + b*x0 + c;             //计算最值

e = string(x0);                     //将对称轴的值符号化
d = string(y0);                     //将最值符号化
  r = sqrt(b^2 - 4*a*c);            //计算判别式的值
  x1 = (-b-r)/(2*a);                //计算较小的根
  x2 = (-b+r)/(2*a);                //计算较大的根

g = string(x1);
h = string(x2);                     //将两根符号化
E = messagebox(['计算该函数的对称轴 x0 和最值 y0 的值及两根 x1 和 x2 的值:';e;d;g;h;],
['退出','绘图'])                    //输出


  x0 = -(b/(2*a));
  y0 = a*x0^2 + b*x0 + c;
  x = -b/(2*a) - u:0.05:-b/(2*a) + u;      //给出 x 数组,确定范围及取点密度
  y = a.*x.^2 + b.*x + c;                  //计算每一个 x 所对应的 y 值
  plot2d(x,y,axesflag = 5);                //绘制函数的图像

  xset('color',21);
  xset('font size',2);                     //设置坐标轴的颜色及大小

  e1 = string(a);                          //将 a 的值符号化
  h1 = string(b);                          //将 b 的值符号化
  g1 = string(c);                          //将 c 的值符号化
  t1 = string(x0);                         //将 x0 的值符号化
  n1 = string(y0);                         //将 y0 的值符号化
  xtitle('y = ' + e1 + '*x^2 + ' + h1 + '*x + ' + g1 + '','duichengzhou:x = ' + t1 + '','y(min) =
' + n1 + '');  //在坐标轴上显示表达式、对称轴及最值
  x0 = -(b/(2*a));
  y0 = a*x0^2 + b*x0 + c;
  e = string(x0);
```

```
  d = string(y0);
  r = sqrt(b^2 - 4 * a * c);
  x1 = ( - b - r)/(2 * a);
  x2 = ( - b + r)/(2 * a);
g = string(x1);
h = string(x2);
E = messagebox(['输出该函数的对称轴 x0 和最值 y0 的值及两根 x1 和 x2 的值:';e;d;g;h;],
['完成'])     //在作出图像的同时利用对话框输出该函数的对称轴 x0 和最值 y0 的值及两根
x1 和 x2 的值
endfunction
```

◆ **程序运行结果**

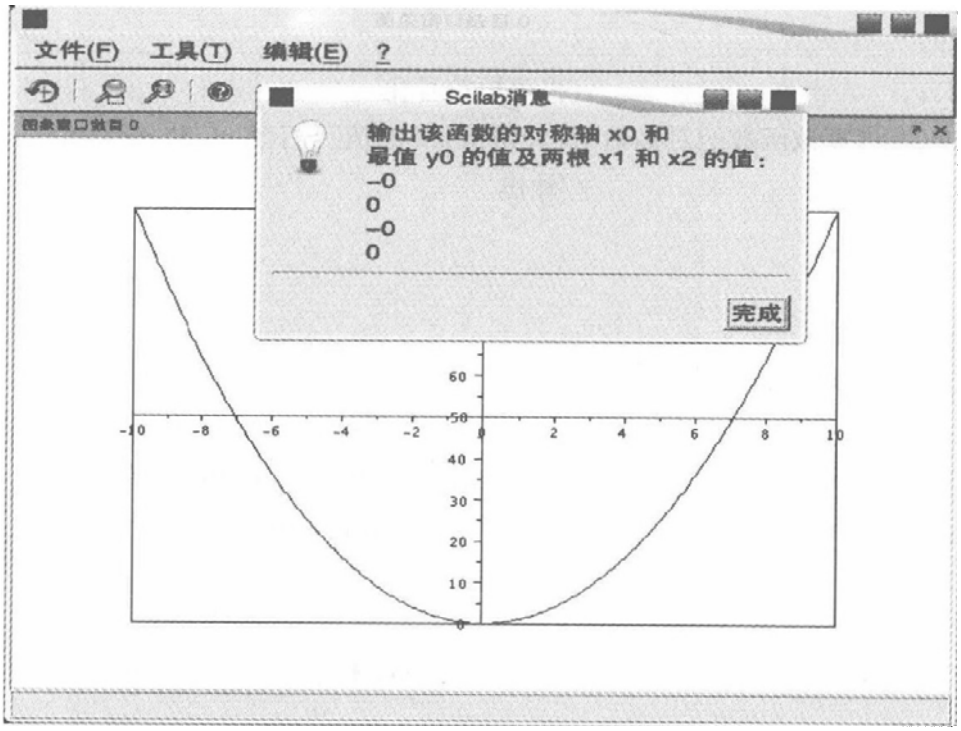


图 6.4 例(2)的执行结果

6.2 排列及组合的计算

(1) 计算排列 A_n^m 的值。

◆ **建模**

该算法主要用于判断 m、n 之间的大小并计算排列 A_n^m 的值。

◆ **流程图**

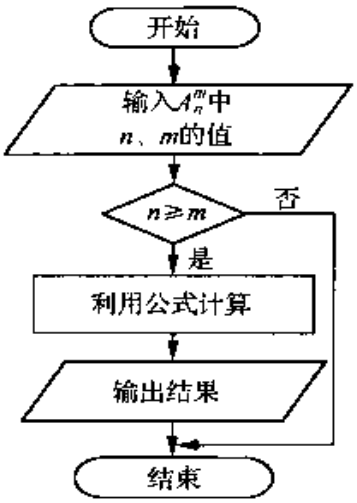


图 6.5 例(1)的流程图

◆ **Scilab 程序**

```
label1 = ['n :';'m :';];
B = x_mdialog([' 请输入 A(n,m)中 n 和 m 的值 : '],label1,['8';'7';]);
n = evstr(B(1));
m = evstr(B(2));   //利用对话框来输入 n、m 的值
D = messagebox([' 确认要进行排列的计算? ';
                '';
                '';
                ''],"modal", "info",["计算"   "退出"]);  //询问是否计算
if n>=m
        s = 1;
    for i = 1:(n - 1)
        i = i + 1;
        s = s * i;
    end   //利用 for 循环求出 s
        t = 1;
    for j = 1:(n - m - 1)
        j = j + 1;
        t = t * j;
    end  //利用 for 循环求出 t
        A(n,m) = s/t;   //求出 A(n,m)
  a = string(A(n,m));  //将 A(n,m)的值符号化
```

```
  E = messagebox(['输出排列 A(n,m) 的值:';a;],"modal", "info",['完成']);//利用对话框
  来输出排列 A(n,m) 的值
elseif n = = m
  s = 1;
    for i = 1 : (n - 1)
        i = i + 1;
        s = s * i;
    end  //利用 for 循环求出 s
         t = 1;
    for j = 1 : (n - m - 1)
        j = j + 1;
        t = t * j;
    end  //利用 for 循环求出 t
         A(n,m) = s/t;  //求出 A(n,m)
  a = string(A(n,m));  //将 A(n,m)的值符号化
  E = messagebox(['输出排列 A(n,m) 的值:';a;],"modal", "info",['完成']);  //利用对话
  框来输出排列 A(n,m) 的值
  else
    E = messagebox(['下标 n  小于上标  m  啦 !!! ';],"modal", "info",['完成'])
  end
```

◆ **程序运行结果**

$n = 8, m = 7$ 时的结果：

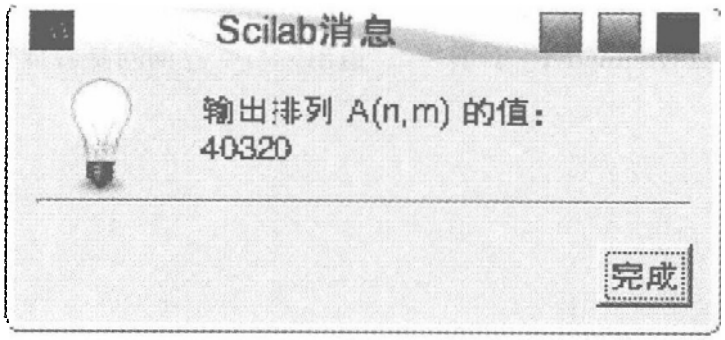


图 6.6　例(1)的执行结果

(2) 计算排列 A_n^m 的值。

◆ **建模**

该算法主要用于计算排列 A_n^m 的值，该算法的优点在于输入 n、m 的值以后不需要判断直接进行计算。

◆ 流程图

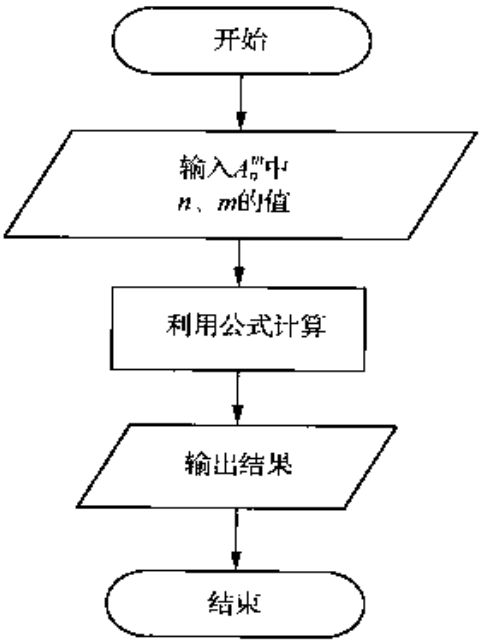


图 6.7 例(2)的流程图

◆ Scilab 程序

```
label1 = ['n :';'m :';];
B = x_mdialog(['请输入 n 和 m 的值,且 n > m :'],label1,['8';'5';]);
n = evstr(B(1));
m = evstr(B(2));      //利用对话框来输入 n,m 的值
D = messagebox(['确认要进行排列的计算?';
                '';
                '';
                ''],"modal", "info",['计算','退出']);  //询问是否计算
k = 1;
  for i = n : -1 : (n-m+1)
    k = k * i;
end  //利用公式计算 A(n,m) 的值
a = string(k);
E = messagebox(['输出排列 A(n,m) 的值:';a;],"modal", "info",['完成']) //利用对话框来输出结果
```

◆ 程序运行结果

$n = 8, m = 5$ 时的结果：

图 6.8 例(2)的执行结果

(3) 计算组合 C_n^m 的值。

◆ **建模**

该算法主要用于计算组合 C_n^m 的值。

◆ **流程图**

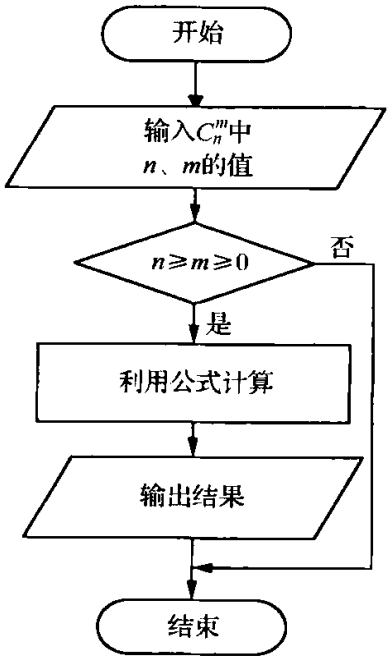


图 6.9 例(3)的流程图

◆ **Scilab 程序**

```
label1 = ['n : ';'m : ';];
B = x_mdialog(['请输入 n 和 m 的值;';'其中 n > m 或 n = m ;';'m 可以等于 0'],
label1,['8';'7';]);
n = evstr(B(1));
m = evstr(B(2));     //利用对话框来输入 n、m 的值
D = messagebox(['确认要进行组合数的计算？';
                '';
                '';
```

```
            '';
            '"],"modal", "info",['计算','退出']);  //询问是否计算
if m = = 0
   k = 1;
   b = string(k);
E = messagebox['输出排列 C (n,m) 的值:';b;],"modal", "info",['完成'])  // C_n^0 的值
  elseif n>m|n = m      // n ≥ m 时计算出 C_n^m 的值
     b = 1,c = 1;
     for i = n : -1 : (n-m+1)
       b = b * i;
     end  //利用 for 循环求出 b
     for i = m : -1 : 1
       c = c * i;
     end        //利用 for 循环求出 c
  C(n,m) = b/c;     //求出 C_n^m 的值
a = string(C(n,m));  //将 C_n^m 的值符号化
E = messagebox['输出排列 C (n,m) 的值:';a;],"modal", "info",['完成'])  //利用对话框输出结果
else
E = messagebox[' 下标 n  小于上标  m  啦 !!! ';],"modal", "info",['完成'])
end
```

◆ **程序运行结果**

$n = 8, m = 7$ 时的结果：

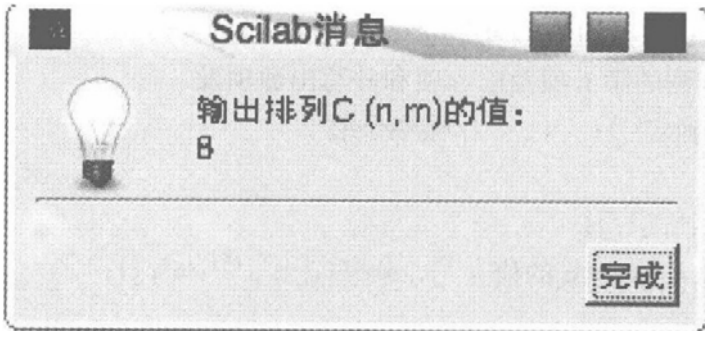


图 6.10 例(3)的执行结果

6.3 数列中的计算

(1) 计算等差数列的第 n 项及前 n 项和 S_n 的值。

◆ **建模**

该算法主要用于在已知某等差数列的首项 a_1、公差 d、项数 n 的情况下，计算等差数列的第 n 项及前 n 项的和。

◆ **流程图**

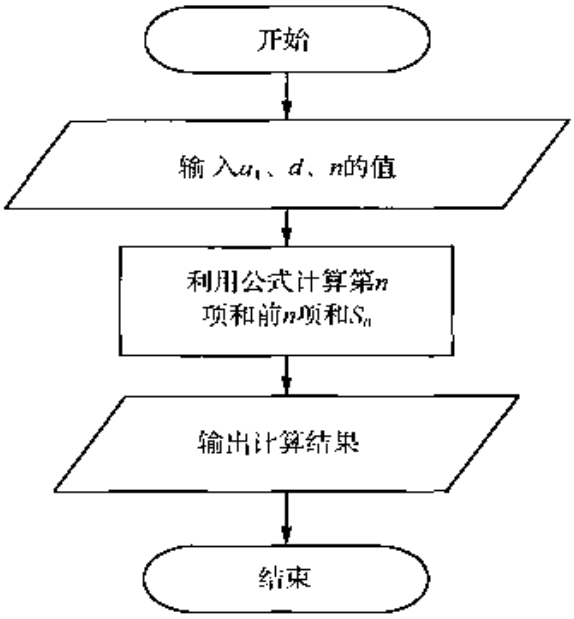


图 6.11 例(1)的流程图

◆ **Scilab 程序**

```
function dengchashulie()    //定义函数名
clc
clear                                    //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';'程序：计算';
'描述:';'计算等差数列的第 n 项及前 n 项和并写出通项及
前 n 项和公式';],['确定']);          //程序介绍

label1 = ['a1 :';'d :';'n :';];
B = x_mdialog(['请输入 a1、d、n 的值 :'],label1,['1';'2';'6';]);
a1 = evstr(B(1));
d = evstr(B(2));
n = evstr(B(3));      //输入设置
if B == []
quit
end        //若选择取消按钮
D = messagebox(['确认计算等差数列的第 n 项及前 n 项和并写出通项及前 n 项和公式?';],
['退出','计算']);    //询问是否进行计算
```

```
//编程
    an = a1 + (n - 1) * d;          //利用公式求出第 n 项
    sn = n * a1 + n * (n - 1) * d/2;      //利用公式求出前 n 项和
    n1 = string(n);          //将 n 进行符号化
    a = string(a1);        //将 a1 进行符号化
    d1 = string(d);        //将 d 进行符号化
    c = a1 - d;
    c1 = string(c);          //将 c 进行符号化
    e = d/2;
    e1 = string(e);          //将 e 进行符号化
    f = a1 - e;
    f1 = string(f);            //将 f 进行符号化
    an = a1 + (n - 1) * d;       //利用公式求出第 n 项
sn = n * a1 + n * (n - 1) * d/2;    //利用公式求出前 n 项和
p = string(an);      //将 an 进行符号化
q = string(sn)       //将 sn 进行符号化
E = messagebox(['输出该等差数列的第 n 项 an 及前 n 项和 Sn 的值:';p;q;],['完成'])   //输出结果
endfunction
```

◆ **程序运行结果**

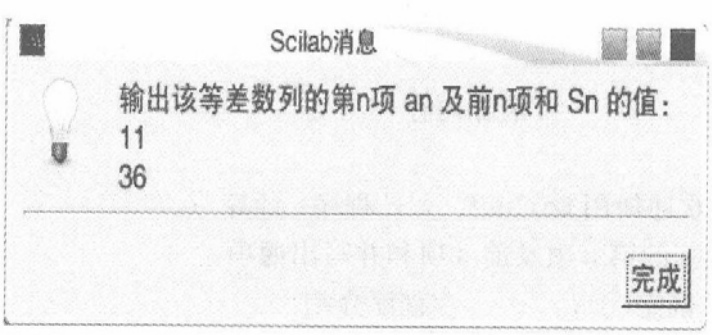


图 6.12　例(1)的执行结果

(2) 计算等比数列的第 n 项及前 n 项的和。

◆ **建模**

该算法主要用于在已知某等比数列的首项 a_1、公比 q、项数 n 的情况下，计算等比数列的第 n 项及前 n 项的和。

◆ **流程图**

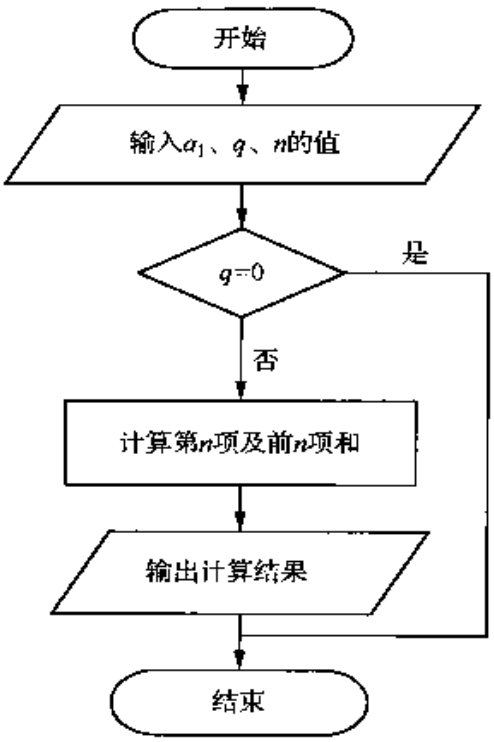


图 6.13 例(2)的流程图

◆ **Scilab 程序**

```
function dengbishulie()          //定义函数名
Clc
clear                            //清除内存
//程序介绍
into = messagebox(['欢迎使用 Scilab! ^_^';'程序：计算';
'描述:';'计算等比数列的第 n 项及前 n 项和并写出通项
及前 n 项和公式'],['确定']);          //程序介绍
输入设置
label1 = ['a1 :';'q :';'n :';];
B = x_mdialog(['请输入 a1、q、n 的值 ：'],label1,['1';'2';'6';]);
a1 = evstr(B(1));
q = evstr(B(2));
n = evstr(B(3));              //输入设置
if B == []
quit
end                  //若选择取消按钮
```

```
D = messagebox(['确认计算等比数列的第 n 项及前 n 项和并写出通项及前 n 项和公式?'],['退出','计算']);          //询问是否进行计算
//编程
if(q==0),break;
  E = messagebox(['q=0 了!!!';],['完成'])  //公比为 0 就停止
else
  bn = a1*(q^(n-1));          //公比不为 0 就利用公式计算出数列的第 n 项
end
  if(q==1)
    sn = n*a1;                //公比为 1 时的特殊情况
    else
  sn = a1*(1-q^n)/(1-q);          //利用公式计算出数列的前 n 项和
  end
e = string(bn);          //将 bn 进行符号化
d = string(sn)           //将 sn 进行符号化
E = messagebox(['输出该等比数列的第 n 项 bn 及前 n 项和 Sn 的值:';e;d;],['完成']) //输出结果
endfunction
```

◆ **程序运行结果**

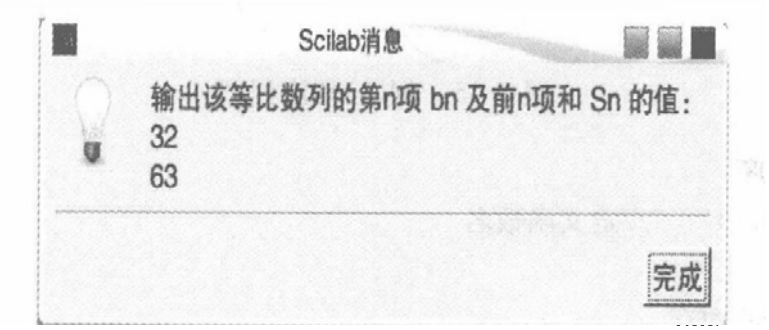


图 6.14 例(2)的执行结果

6.4 三角函数中的计算与绘图

(1) 计算正弦函数 $y = A\sin(Wx + Q) + H$ 的周期、最值并绘制图像。

◆ **建模**

该算法主要用于计算正弦函数 $y = A\sin(Wx + Q) + H$ 的周期、最值并绘制其图像。A、W、Q、H 分别为正弦函数 $y = A\sin(Wx + Q) + H$ 中的系数。

◆ 流程图

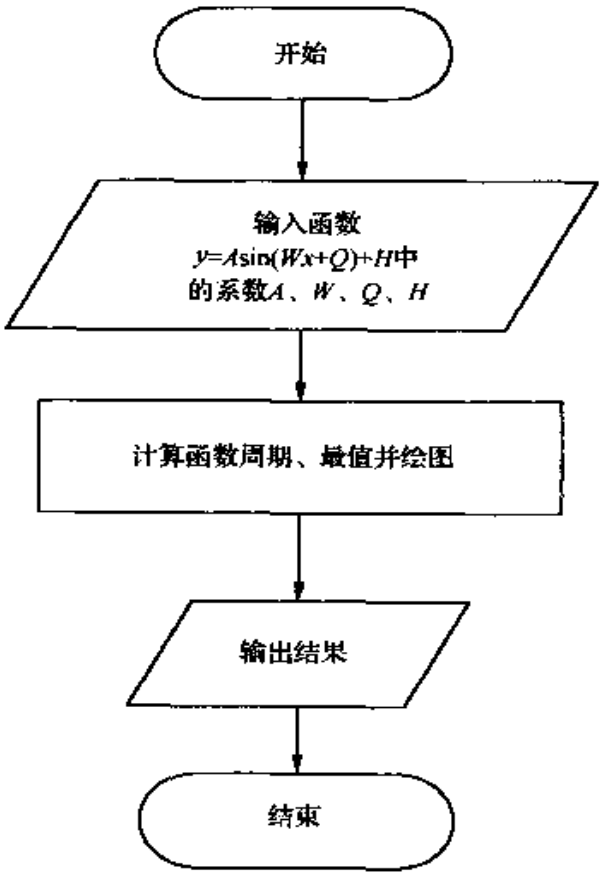


图 6.15 例(1)的流程图

◆ Scilab 程序

```
function zheng_xian()      //定义函数名
clc
clear          //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';
 '程序:绘制图形';'描述:';'计算正弦函数
y = A * sin(W * x + Q) + H 的最值、周期并作图形';''],
['确定']);      //程序功能介绍
label1 = ['A :';'W :';'Q :';'H';];
B = x_mdialog(['请输入 y = A * sin(W * x + Q) + H 的系数 :'],...
label1,['1';'1';'0';'0']);
A = evstr(B(1));
W = evstr(B(2));
Q = evstr(B(3));
H = evstr(B(4));      //输入设置
if B = = []
```

```
quit
end  //若选择取消按钮
D = messagebox(['确认计算正弦函数 y = A * sin(W * x + Q) + H
的最大值、最小值、周期?'],['退出','计算']);  //输入参数
//编程
  T = 2 * %pi/W;          //计算正弦函数的周期
  t = string(T);          //将 T 的值符号化
  x = -(2 * %pi)/W : 0.01 : (2 * %pi)/W;  //计算对称轴
  y = A. * sin(W. * x + Q) + H;           //计算最值
  MAX = max(y);           //将最大值赋给 MAX
  m = string(MAX);        //将 MAX 符号化
  MIN = min(y);           //将最小值赋给 MIN
  n = string(MIN);        //将 MIN 符号化
E = messagebox(['输出 y 的最大值 y(max) 和最小值 y(min) 以及周期 T 的值:';m;n;t;],['退
出','作图'])              //输出最大值、最小值及周期 T 的值
  x = -(2 * %pi)/W : 0.01 : (2 * %pi)/W;    //给定 x 数组,确定范围及取点密度
  y = A. * sin(W. * x + Q) + H;             //计算出每个 x 所对应的 y 值
  T = 2 * %pi/W;
  plot2d(x,y,rect = [-T-2,-2-A-H,T+2,2+A+H],axesflag = 5);  //绘制函数图像
  xset('color',21);       //设定坐标轴的颜色
  xset('font size',2);    //设定坐标轴的长度
  a1 = string(A);
  b1 = string(W);
  c1 = string(Q);
  d1 = string(H);
  xtitle('y = ' + a1 + ' * sin(' + b1 + ' * x + ' + c1 + ') + ' + d1 + '');  //在坐标轴上输出函数表
  达式
x = -(2 * %pi)/W : 0.01 : (2 * %pi)/W;
y = A. * sin(W. * x + Q) + H;
  T = 2 * %pi/W;
  t = string(T);
  MAX = max(y);
  m = string(MAX);
  MIN = min(y);
  n = string(MIN);
E = messagebox(['输出 y 的最大值 y(max) 和最小值 y(min) 以及周期 T 的值:';m;n;t;],['完
成'])          //输出设置
endfunction
```

◆ **程序运行结果**

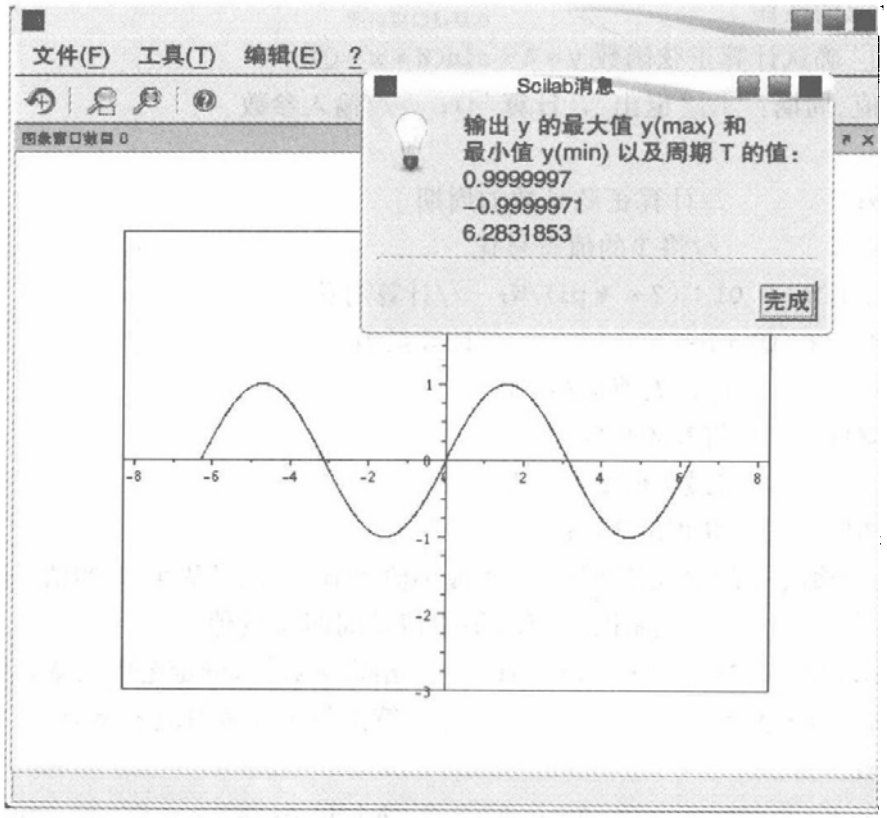


图 6.16 例(1)的执行结果

(2) 计算余弦函数 $y = A\cos(Wx + Q) + H$ 的周期、最值并绘制图像。

◆ **建模**

该算法主要用于计算余弦函数 $y = A\cos(Wx + Q) + H$ 的周期、最值并绘制图像。A、W、Q、H 分别为余弦函数 $y = A\cos(Wx + Q) + H$ 中的系数。

◆ **流程图**(图 6.17)

◆ **Scilab 程序**

```
function yu_xian()      //定义函数名
clc
clear                   //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';
'程序: 绘制图形 ';' 描述:'; '计算余弦函数
y = A * cos(W * x + Q) + H 的最值、周期并作图形 ';
''],[' 确定 ']);        //程序功能介绍
label1 = ['A :';'W :';'Q :';'H';];
B = x_mdialog(['请输入 y = A * cos(W * x + Q) + H 的系数 :'],...
label1,['1';'1';'0';'0']);
A = evstr(B(1));
W = evstr(B(2));
```

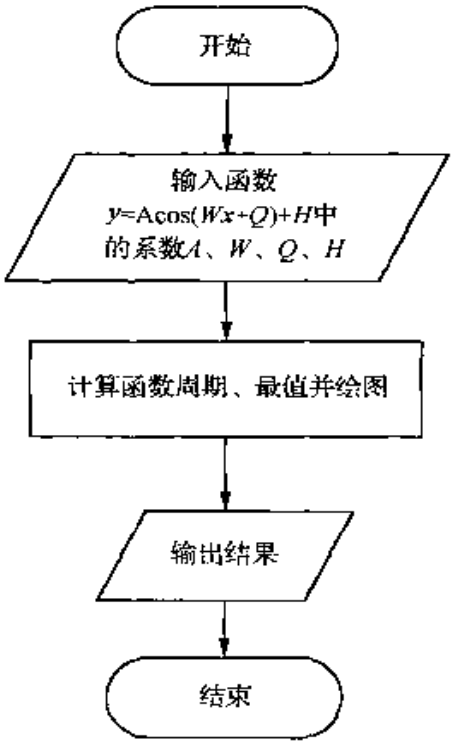


图 6.17 例(2)的流程图

```
Q = evstr(B(3));
H = evstr(B(4));   //输入设置
if B == []
quit
end        //若选择取消按钮
D = messagebox(['确认计算余弦函数 y = A * cos(W * x + Q) + H 的最大值、最小值、周期?'],['退出','计算']);      //输入参数
//编程
  x = -(2 * %pi)/W : 0.01 : (2 * %pi)/W;    //给定 x 数组,确定范围及取点密度
  y = A. * cos(W. * x + Q) + H;              //计算出每个 x 所对应的 y 值
  T = 2 * %pi/W;                            //计算余弦函数的周期
  t = string(T);                            //将 T 的值符号化
  MAX = max(y);                             //将函数的最大值赋给 MAX
  m = string(MAX);                          //将 MAX 的值符号化
  MIN = min(y);                             //将函数的最小值赋给 MIN
  n = string(MIN);                          //将 MIN 的值符号化
  E = messagebox(['输出 y 的最大值 y(max) 和最小值 y(min) 以及周期 T 的值:';m;n;t;],
  ['退出','绘图'])        //输出最大值、最小值及周期 T 的值
  x = -(2 * %pi)/W : 0.01 : (2 * %pi)/W;
  y = A. * cos(W. * x + Q) + H;
  T = 2 * %pi/W;
```

```
plot2d(x,y,rect=[-T-2,-2-A-H,T+2,2+A+H],axesflag=5);   //绘制余弦函数的图像
xset('color',21);        //设定坐标轴的颜色
xset('font size',3);        //设定坐标轴的长度
a1=string(A);
b1=string(W);
c1=string(Q);
d1=string(H);
xtitle('y='+a1+'*cos('+b1+'*x+'+c1+')+'+d1+'');      //在坐标轴上输出表达式
x=-(2*%pi)/W:0.01:(2*%pi)/W;
y=A.*cos(W.*x+Q)+H;
T=2*%pi/W;
t=string(T);
MAX=max(y);
m=string(MAX);
MIN=min(y);
n=string(MIN);
E=messagebox(['输出 y 的最大值 y(max) 和最小值 y(min) 以及周期 T 的值:';m;n;t;],['完成'])          //输出结果
endfunction
```

◆ **程序运行结果**

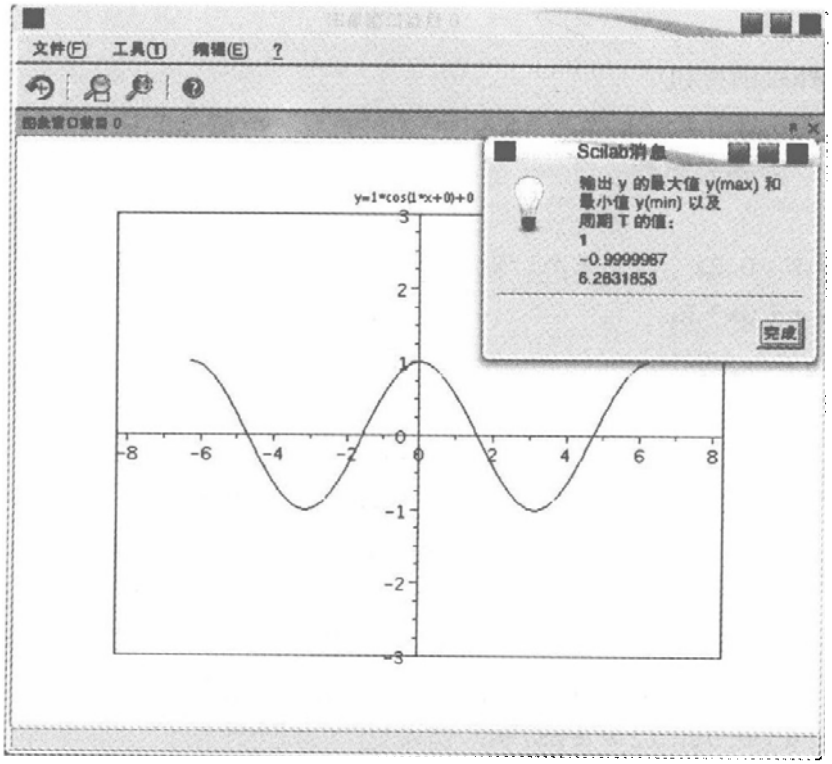


图 6.18 例(2)的执行结果

(3) 计算正切函数 $y = A\tan(Wx + Q) + H$ 的周期、最值并绘制图像。

◆ **建模**

该算法主要用于计算正切函数 $y = A\tan(Wx + Q) + H$ 的周期、最值并绘制图像。

A、W、Q、H 分别为正切函数 $y = A\tan(Wx + Q) + H$ 中的系数。

◆ **流程图**

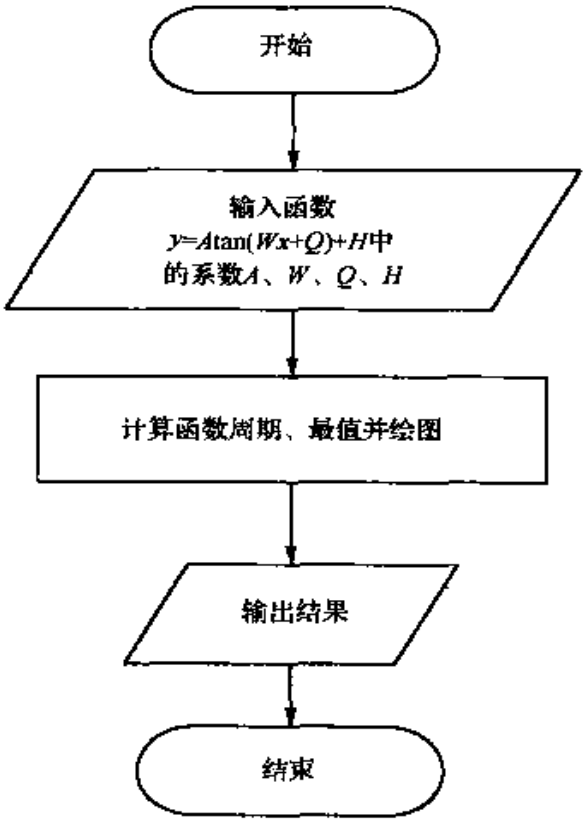


图 6.19 例(3)的流程图

◆ **Scilab 程序**

```
function zheng_qie()  //定义函数名
clc
clear                 //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';
'程序：绘制图形 '; '描述：';'计算正切函数
y = A * tan(W * x + Q) + H 的周期并作图 '],
['确定 ']);  //程序功能介绍
label1 = ['A ：';'W ：';'Q ：';'H';];
B = x_mdialog(['请输入 y = A * tan(W * x + Q) + H 的系数 ： '],...
label1,['1';'1';'0';'0']);
```

```
A = evstr(B(1));
W = evstr(B(2));
Q = evstr(B(3));
H = evstr(B(4));   //输入设置
if B = = []
quit
end          //若选择取消按钮
输入参数


D = messagebox(['确认计算正切函数 y = A * tan(W * x + Q) + H 的周期? '],['退出 ','计算周期 ']);
//编程
  T = %pi/W;                              //计算正切函数的周期
  t = string(T);                          //将 T 的值符号化
  E = messagebox(['输出周期 T 的值:';t;],['退出 ','绘图 ']);
  x = -2 * (%pi)/W : 0.01 : 2 * (%pi)/W;          //给定 x 数组,确定范围及取点密度
  y = A. * tan(W. * x + Q) + H;                   //计算出每个 x 所对应的 y 值
  plot2d(x,y,rect = [-T-2, -2-A-H,T+2,2+A+H],axesflag = 5); //绘制函数图像
  xset('color',21);            //设定坐标轴的颜色
  xset('font size',3);          //设定坐标轴的长度
  xset('pattern',21);
  t1 = string(T);
  a1 = string(A);
  b1 = string(W);
  c1 = string(Q);
  d1 = string(H);
  xtitle('y = ' + a1 + ' * tan(' + b1 + ' * x + ' + c1 + ') + ' + d1 + '');     //在坐标轴上输出函数
  表达式
T = %pi/W;              //计算函数的周期
t = string(T);          //将 T 的值符号化
E = messagebox(['输出周期 T 的值:';t;],['完成 '])                    //输出结果
endfunction
```

◆ **程序运行结果**

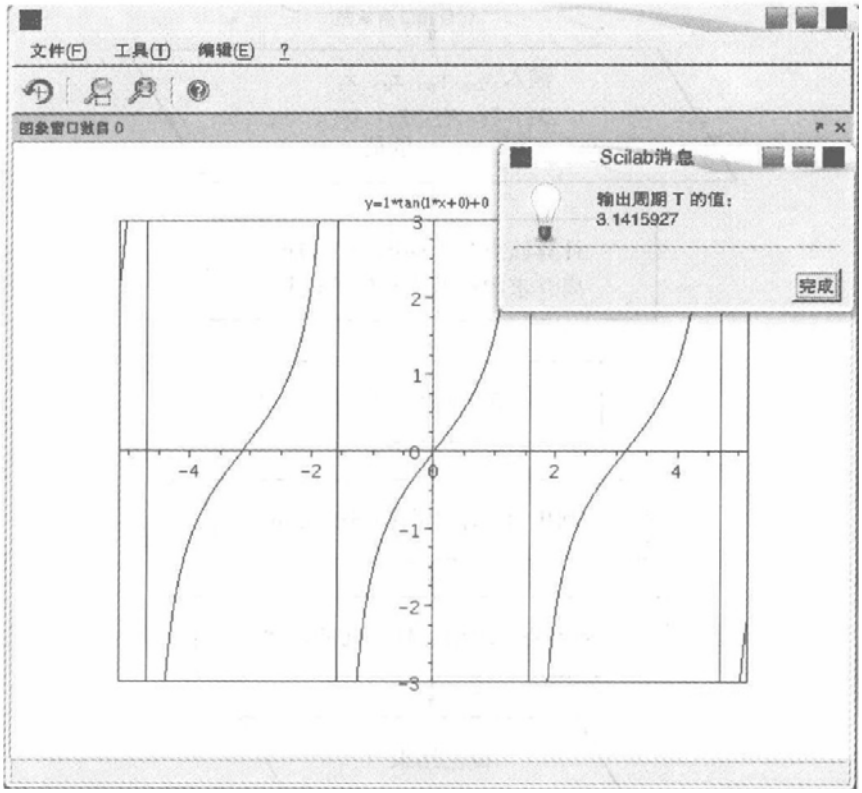


图 6.20　例(3)的执行结果

6.5　立体几何中的计算

(1) 求空间某一点到某平面的距离。

◆ **建模**

该算法主要用于在已知空间某一点的坐标及某平面上的三个点的坐标，求该点到平面的距离。其中 (x_0, y_0, z_0) 为空间上某一点的坐标，(x_1, y_1, z_1)，(x_2, y_2, z_2)，(x_3, y_3, z_3) 为某平面上的三个点的坐标。

◆ **流程图**(图 6.21)

◆ **Scilab 程序**

```
label1 = ['x0:';'y0:';'z0:';'x1:';'y1:';'z1:';'x2:';'y2:';'z2:';'x3:';'y3:';'z3:';];
B = x_mdialog([' 请输入各点的坐标 : '],label1,['0';'0';'1';'3';'6';'9';'3';'7';...'6';'3';
'8';'1';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
z0 = evstr(B(3));
x1 = evstr(B(4));
y1 = evstr(B(5));
```

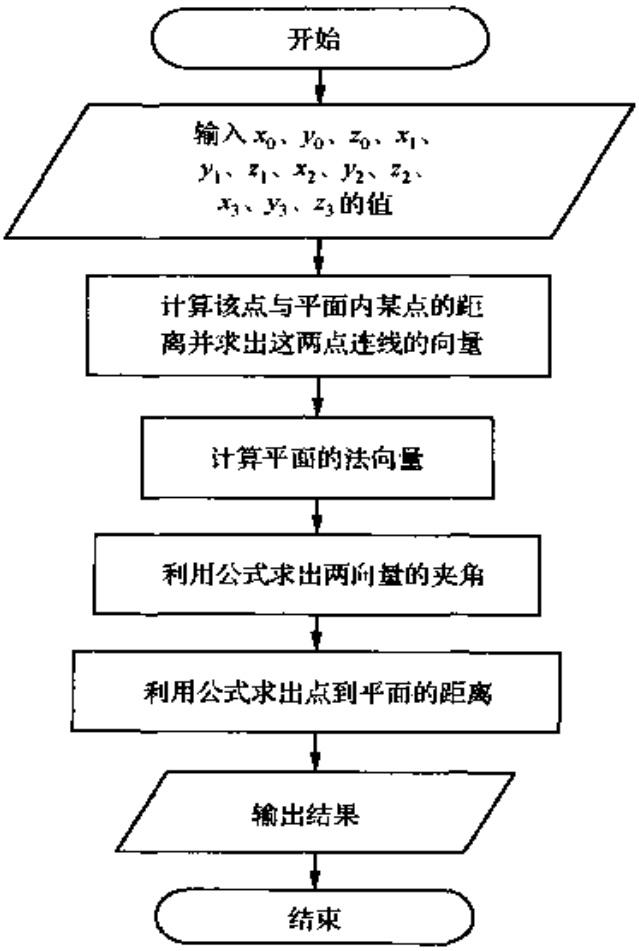


图 6.21 例(1)的流程图

```
z1 = evstr(B(6));
x2 = evstr(B(7));
y2 = evstr(B(8));
z2 = evstr(B(9));
x3 = evstr(B(10));
y3 = evstr(B(11));
z3 = evstr(B(12));   //利用对话框来输入 x0、y0、z0、x1、y1、z1、x2、y2、z2、x3、y3、z3 的值
D = messagebox(['确认进行点到平面的距离的计算?';
                '';
                '';
                ''],"modal", "info",['计算','退出']);  //询问是否计算
d = ((x0 - x1)^2 + (y0 - y1)^2 + (z0 - z1)^2)^(1/2);  //求出该点到平面内某一点(x1、y1、z1)的距离
  a = x0 - x1;
  b = y0 - y1;
  c = z0 - z1;
```

```
n=[a,b,c];   //求出点(x0、y0、z0)与点(x1、y1、z1)所在直线的向量
a1=((y3-y2)*(z2-z1)-(y2-y1)*(z3-z2));
b1=((z3-z2)*(x2-x1)-(x3-x2)*(z2-z1));
c1=((x3-x2)*(y2-y1)-(x2-x1)*(y3-y2));
m=[a1,b1,c1]; //求出平面的法向量
costh=abs((a*a1+b*b1+c*c1)/((a^2+b^2+c^2)^(1/2)*(a1^2+b1^2+c1^2)^(1/2)));
//利用公式计算两向量夹角的余弦值的绝对值
h=d*costh;     //求出点到平面的距离
m=string(h);  //将h的值符号化
E=messagebox(['输出距离h的值:';m;],"modal", "info",['完成']) //利用对话框输出结果
```

◆ **程序运行结果**

$x_0=0, y_0=0, z_0=1, x_1=3, y_1=6, z_1=9, x_2=3, y_2=7, z_2=6, x_3=3, y_3=8, z_3=1$ 时的结果：

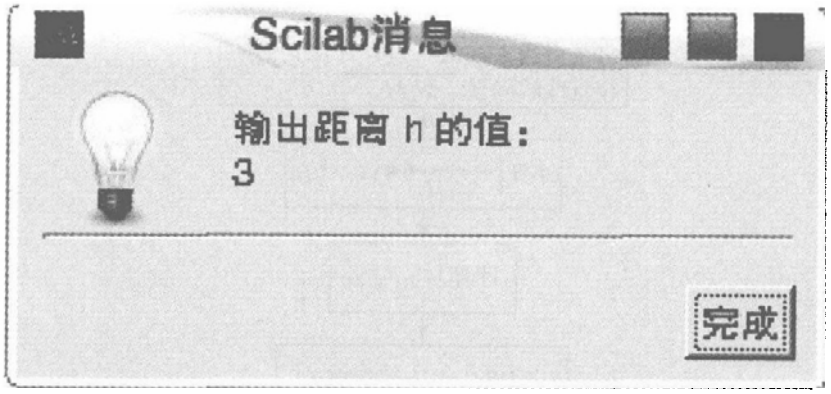


图 6.22 例(1)的执行结果

(2) 求两条异面直线的距离。

◆ **建模**

该算法主要用于在已知空间两条异面直线上的四个点的坐标，求这两条异面直线的距离。算法的数学理论依据是：若两条直线 l_1 与 l_2 的方程分别为：$l_1:\frac{x-x_1}{X_1}=\frac{y-y_1}{Y_1}=\frac{z-z_1}{Z_1}$ 和 $l_2:\frac{x-x_2}{X_2}=\frac{y-y_2}{Y_2}=\frac{z-z_2}{Z_2}$. 这里直线 l_1 是由点 $M_1(x_1,y_1,z_1)$ 与向量 $v_1=\{X_1,Y_1,Z_1\}$ 决定的，l_2 是由点 $M_2(x_2,y_2,z_2)$ 与向量 $v_2=(X_2,Y_2,Z_2)$ 决定的。两异面直线间的距离 d 就是他们共垂线的长，因为 $|\overrightarrow{M_1M_2}\cdot(v_1\times v_2)|$ 为由三向量 $\overrightarrow{M_1M_2}, v_1, v_2$ 构成的平行六面体的体积，而 $|v_1\times v_2|$ 为由向量 v_1, v_2 构成的平行四边形的面积，也就是上述平行六面体的一个面的面积。容易知道两异面直线间的距离 d 恰为三向量 $\overrightarrow{M_1M_2}, v_1, v_2$ 构成的平行六面体在两向量 v_1, v_2 构成的平行四边形面上的高。则直线 l_1 与 l_2 之间的距离为

$$d=\frac{|(\overrightarrow{M_1M_2},v_1,v_2)|}{|v_1\times v_2|}=\left|\frac{\begin{vmatrix}x_2-x_1 & y_2-y_1 & z_2-z_1\\ X_1 & Y_1 & Z_1\\ X_2 & Y_2 & Z_2\end{vmatrix}}{\sqrt{\begin{vmatrix}Y_1 & Z_1\\ Y_2 & Z_2\end{vmatrix}^2+\begin{vmatrix}Z_1 & X_1\\ Z_2 & X_2\end{vmatrix}^2+\begin{vmatrix}X_1 & Y_1\\ X_2 & Y_2\end{vmatrix}^2}}\right|$$

◆ 流程图

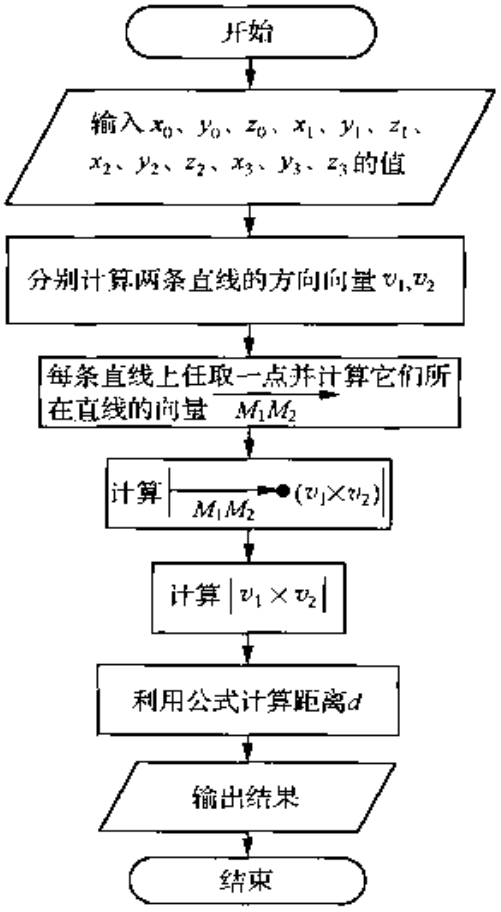


图 6.23 例(2)的流程图

◆ Scilab 程序

```
label1 = ['x0:';'y0:';'z0';'x1 :';'y1 :';'z1:';'x2 :';'y2:';'z2:';'x3:';'y3:';'z3:';];
B = x_mdialog(['请输入各点的坐标 :';'( x0,y0,z0 ) 、( x1 ,y1,z1 ) 为其中一条直线上的两个点 ';'( x2,y2,z2 )、( x3,y3,z3 )为另一条直线上的两个点:'],label1,['0';'0';'1';'3';'6';'9';'3';'7';'6';'3';'8';'1';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
z0 = evstr(B(3));
x1 = evstr(B(4));
y1 = evstr(B(5));
z1 = evstr(B(6));
```

```
x2 = evstr(B(7));
y2 = evstr(B(8));
z2 = evstr(B(9));
x3 = evstr(B(10));
y3 = evstr(B(11));
z3 = evstr(B(12));  //利用对话框来输入 x0、y0、z0、x1、y1、z1、x2、y2、z2、x3、y3、z3 的值
D = messagebox(['确认进行空间异面直线的距离的计算?';
               '';
               '';
               '';
               ''],"modal", "info",['计算','退出']);  //询问是否计算
  a = x1 - x0;
  b = y1 - y0;
  c = z1 - z0;
  n = [a,b,c]      //求出其中一条直线的方向向量

  a1 = x3 - x2;
  b1 = y3 - y2;
  c1 = z3 - z2;
  m = [a1,b1,c1]  //求出另一条直线的方向向量

  A = [x2 - x1 y2 - y1 z2 - z1;a b c;a1 b1 c1];  //给出矩阵 A

  B = ((b * c1 - c * b1)^2 + (c * a1 - a * c1)^2 + (a * b1 - b * a1)^2)^(1/2); //求出 |v1 × v2|
  d = abs(det(A)/B);  //利用公式求出距离
  m = string(d);
E = messagebox(['输出空间异面直线的距离 h  的值:';m;],"modal", "info",['完成'])
```

◆ **程序运行结果**

$x_0 = 0, y_0 = 0, z_0 = 1, x_1 = 3, y_1 = 6, z_1 = 9, x_2 = 3, y_2 = 7, z_2 = 6, x_3 = 3, y_3 = 8, z_3 = 1$ 时的结果：

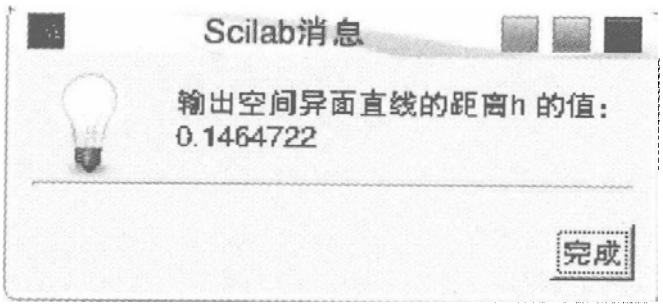


图 6.24 例(2)的执行结果

(3) 计算直线与直线夹角。

◆ **建模**

该算法主要用在分别已知两条直线上的两点的坐标,计算两条直线的夹角。$(x_0,y_0,z_0),(x_1,y_1,z_1),(x_2,y_2,z_2),(x_3,y_3,z_3)$ 分别为两条直线上的四个点的坐标。

◆ **流程图**

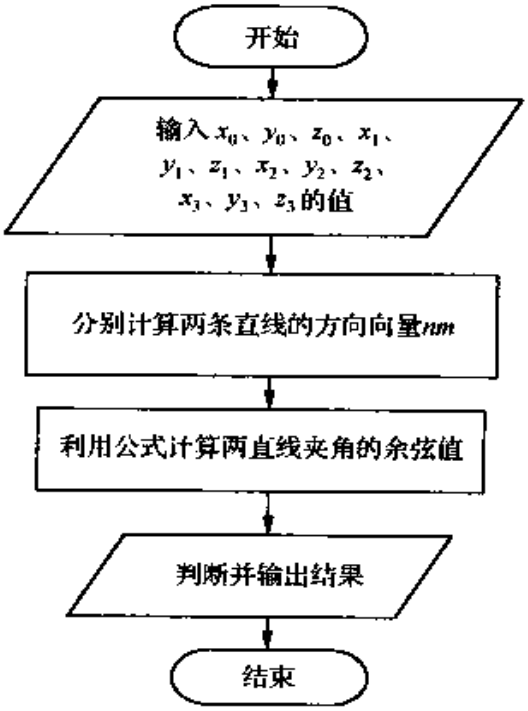


图 6.25 例(3)的流程图

◆ **Scilab 程序**

```
label1 = ['x0:';'y0:';'z0';'x1 :';'y1 :';'z1:';'x2 :';'y2:';'z2:';'x3:';'y3:';'z3:';];
B = x_mdialog(['请输入各点的坐标 :';'(x0,y0,z0),(x1,y1,z1)为其中一条直线上的两个点';'(x2,y2,z2),(x3,y3,z3)为另一条直线上两个点的坐标。'],label1,['0';'0';'1';'3';'6';'9';'3';'7';'6';'3';'8';'1';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
z0 = evstr(B(3));
x1 = evstr(B(4));
y1 = evstr(B(5));
z1 = evstr(B(6));
x2 = evstr(B(7));
y2 = evstr(B(8));
z2 = evstr(B(9));
```

```
x3 = evstr(B(10));
y3 = evstr(B(11));
z3 = evstr(B(12));  //利用对话框来输入 x0、y0、z0、x1、y1、z1、x2、y2、z2、x3、y3、z3 的值
D = messagebox(['确认进行空间直线与直线夹角的求解?';
                '';
                '';
                ''],"modal", "info",['计算','退出']);   //询问是否计算
a = x1 - x0;
b = y1 - y0;
c = z1 - z0;
n = [a,b,c];  //计算其中一条直线的方向向量
a1 = x3 - x2;
b1 = y3 - y2;
c1 = z3 - z2;
m = [a1,b1,c1];     //计算另一条直线的方向向量
costh = (a * a1 + b * b1 + c * c1)/((a^2 + b^2 + c^2)^(1/2) * (a1^2 + b1^2 + c1^2)^(1/2)) //利用
公式计算两条直线夹角的余弦值
if costh >= 0
   costh = costh       //若夹角的余弦值大于 0,直接输出
else
  costh = - costh      //若夹角的余弦值小于 0,取相反数
  end
  m = string(costh);
E = messagebox(['输出空间直线与直线的夹角 cos (Θ) 的值:';m;],"modal", "info",['完成'])
```

◆ 程序运行结果

$(x_0, y_0, z_0) = (0,0,1), (x_1, y_1, z_1) = (3,6,9), (x_2, y_2, z_2) = (3,7,6), (x_3, y_3, z_3) = (3,8,1)$ 时的结果：

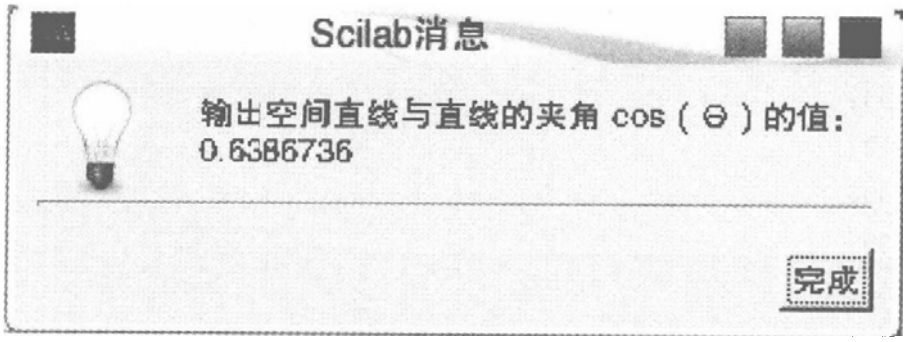


图 6.26 例(3)的执行结果

(4) 计算直线与平面的夹角。

◆ **建模**

该算法主要用于在已知直线上两个点的坐标及某平面上的三个点的坐标，计算直线与该平面的夹角。其中 $(x_0,y_0,z_0),(x_1,y_1,z_1),(x_2,y_2,z_2)$ 为平面上三个点的坐标，$(x_3,y_3,z_3),(x_4,y_4,z_4)$ 为直线上的两个点的坐标。

◆ **流程图**

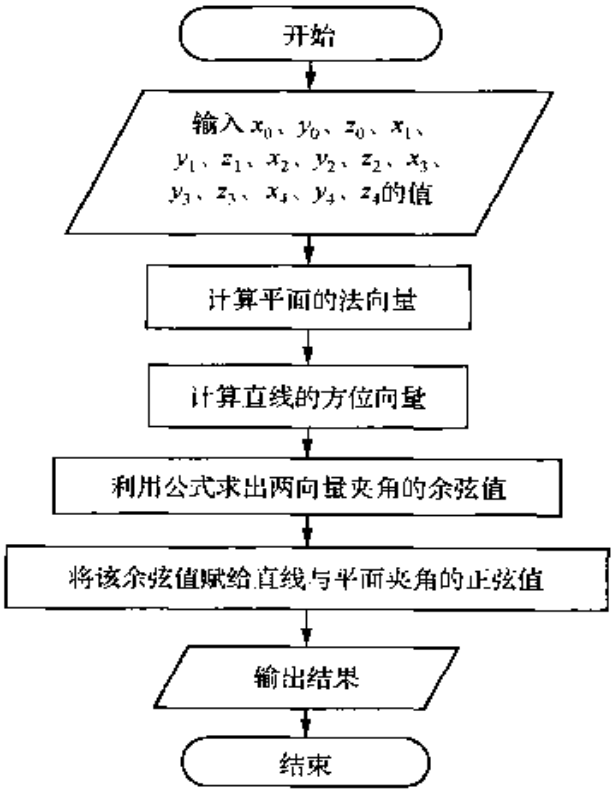


图 6.27 例(4)的流程图

◆ **Scilab 程序**

```
label1 = ['x0:';'y0:';'z0';'x1 :';'y1 :';'z1:';'x2 :';'y2:';'z2:';'x3:';'y3:';'z3:';'x4';'y4';'z4';];
B = x_mdialog(['请输入各点的坐标 :';'(x0,y0,z0),(x1,y1,z1),(x2,y2,z2)为平面上三个点的坐标';'(x3,y3,z3),(x4,y4,z4)为直线上的两个点的坐标';],label1,['0';'0';'1';'3';'6';'9';'3';'7';'6';'3';'8';'1';'3';'7';'5';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
z0 = evstr(B(3));
x1 = evstr(B(4));
y1 = evstr(B(5));
z1 = evstr(B(6));
x2 = evstr(B(7));
y2 = evstr(B(8));
```

```
z2 = evstr(B(9));
x3 = evstr(B(10));
y3 = evstr(B(11));
z3 = evstr(B(12));
x4 = evstr(B(13));
y4 = evstr(B(14));
z4 = evstr(B(15));     //利用对话框来输入 x0、y0、z0、x1、y1、z1、x2、y2、z2、x3、y3、z3、x4、y4、z4 的值

D = messagebox['确认计算直线与平面的夹角?';
               '';
               '';
               '';
               ''],"modal", "info",['计算','退出']);
    a = [x1 - x0,y1 - y0,z1 - z0];  //计算向量 a
    b = [x2 - x1,y2 - y1,z2 - z1];   //计算向量 b
    c = (y1 - y0) * (z2 - z1) - (y2 - y1) * (z1 - z0);
    d = (z1 - z0) * (x2 - x1) - (x1 - x0) * (z2 - z1);
    e = (x1 - x0) * (y2 - y1) - (y1 - y0) * (x2 - x1);
    n = [c,d,e];  //计算平面的法向量
    f = string(c);
    g = string(d);
    h = string(e);  //将 c、d、e 的值发挥很好
    E = messagebox['输出平面的法向量:';f;g;h],"modal", "info",['继续']) //利用对话框输出平面的法向量
    a1 = x4 - x3;
    b1 = y4 - y3;
    c1 = z4 - z3;
    m = [a1,b1,c1];  //计算直线的方位向量
  i = string(a1);
  j = string(b1);
  l = string(c1);  //将 a1、b1、c1 的值符号化
E = messagebox['输出直线的方位向量:';i;j;l],"modal", "info",['继续'])  //利用对话框输出直线的方位向量
    costh = abs((c * a1 + d * b1 + e * c1)/((c^2 + d^2 + e^2)^(1/2) * (a1^2 + b1^2 + c1^2)^(1/2)))  //计算直线的方位向量与平面法向量之间的夹角的余弦值
    sinth = costh;  //该余弦值也为直线与平面夹角的正弦值
    k = string(sinth);  //将 sinth 符号化
```

```
E = messagebox(['输出空间直线与平面的夹角 sin ( Θ ) 的值:';k;],"modal", "info",['完成'])
//利用对话框输出结果
```

◆ **程序运行结果**

输入 $x_0=0, y_0=0, z_0=1, x_1=3, y_1=6, z_1=9, x_2=3, y_2=7, z_2=6, x_3=3, y_3=8, z_3=1, x_4=3, y_4=7, z_4=5$ 时的结果：

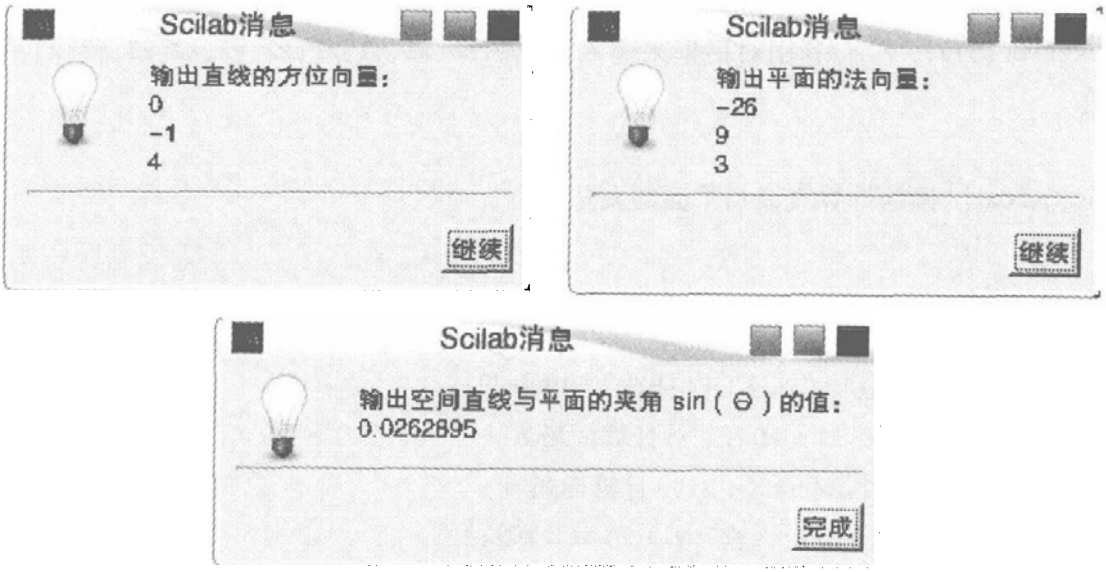


图 6.28 例(4)的执行结果

(5) 计算平面与平面的夹角。

◆ **建模**

该算法主要用于在已知空间平面 α 上三个点 $A(x_0,y_0,z_0), B(x_1,y_1,z_1), C(x_2,y_2,z_2)$ 的坐标及空间平面 β 上三个点 $A_1(x_3,y_3,z_3), B_1(x_4,y_4,z_4), C_1(x_5,y_5,z_5)$ 的坐标，求这两个平面的夹角。

◆ **流程图**

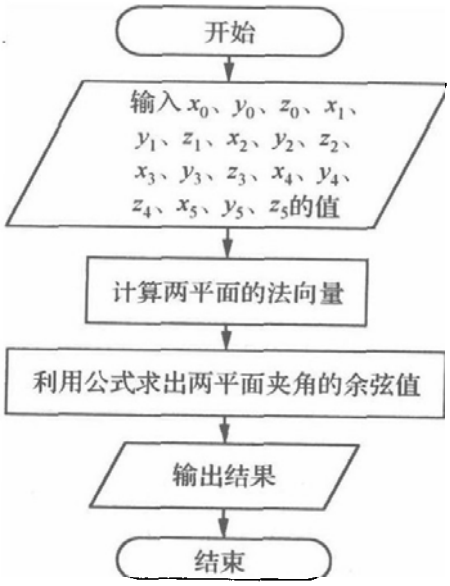


图 6.29 例(5)的流程图

◆ **Scilab 程序**

```
label1 = ['x0:';'y0:';'z0';'x1 :';'y1 :';'z1:';'x2 :';'y2:';'z2:';'x3:';'y3:';'z3:';'x4';'y4';
'z4';'x5';'y5';'z5';];
B = x_mdialog(['请输入各点的坐标 :'], label1,['0';'0';'1';'3';'6';'9';'3';'7';'6';'3';'8';'1';
'3';'6';'9';'0';'6';'5';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
z0 = evstr(B(3));
x1 = evstr(B(4));
y1 = evstr(B(5));
z1 = evstr(B(6));
x2 = evstr(B(7));
y2 = evstr(B(8));
z2 = evstr(B(9));
x3 = evstr(B(10));
y3 = evstr(B(11));
z3 = evstr(B(12));
x4 = evstr(B(13));
y4 = evstr(B(14));
z4 = evstr(B(15));
x5 = evstr(B(16));
y5 = evstr(B(17));
z5 = evstr(B(18));  //利用对话框来输入 x0、y0、z0、x1、y1、z1、x2、y2、z2、x3、y3、z3、x4、y4、z4、
x5、y5、z5 的值
D = messagebox['确认计算平面与平面的夹角?';
                '';
                '';
                ''],"modal", "info",['计算','退出']);  //询问是否计算
AB = [x1 - x0,y1 - y0,z1 - z0];  //求出向量 AB
AC = [x2 - x0,y2 - y0,z2 - z0];  //求出向量 AC
a = (y1 - y0) * (z2 - z0) - (z1 - z0) * (y2 - y0);
b = (z1 - z0) * (x2 - x0) - (x1 - x0) * (z2 - z0);
c = (x1 - x0) * (y2 - y0) - (y1 - y0) * (x2 - x0);
n1 = [a,b,c];   //求出该平面的法向量
a1 = string(a);
b1 = string(b);
c1 = string(c);  //将 a、b、c 的值符号化
```

```
E = messagebox(['输出其中一个平面的法向量的值:';a1;b1;c1],"modal", "info",['继续'])
//利用对话框输出该平面法向量的值

A1B1 = [x4 - x3,y4 - y3,z4 - z3];  //求出向量 A1B1
A1C1 = [x5 - x3,y5 - y3,z5 - z3];  //求出向量 A1C1
d = (y4 - y3) * (z5 - z3) - (z4 - z3) * (y5 - y3);
e = (z4 - z3) * (x5 - x3) - (x4 - x3) * (z5 - z3);
f = (x4 - x3) * (y5 - y3) - (y4 - y3) * (x5 - x3);
n2 = (d,e,f);  //求出该平面的法向量
d1 = string(d);
e1 = string(e);
f1 = string(f);  //将 d、e、f 的值符号化
E = messagebox(['输出另一个平面的法向量的值:';d1;e1;f1],"modal", "info",['继续'])
//利用对话框输出该平面法向量的值
costh = (abs(a * d + b * e + c * f))/(((a^2 + b^2 + c^2) * (d^2 + e^2 + f^2))^1/2); //利用公式计算两平面夹角的余弦值
m = string(costh);  //将 costh 的值符号化
E = messagebox(['输出两平面夹角的余弦值 cos(Θ)的值为:';m;],"modal", "info",['完成'])
//利用对话框输出结果
```

◆ **程序运行结果**

输入 $x_0=0, y_0=0, z_0=1, x_1=3, y_1=6, z_1=9, x_2=3, y_2=7, z_2=6, x_3=3, y_3=8, z_3=1, x_4=3, y_4=6, z_4=9, x_5=0, y_5=6, z_5=5$ 时的结果：

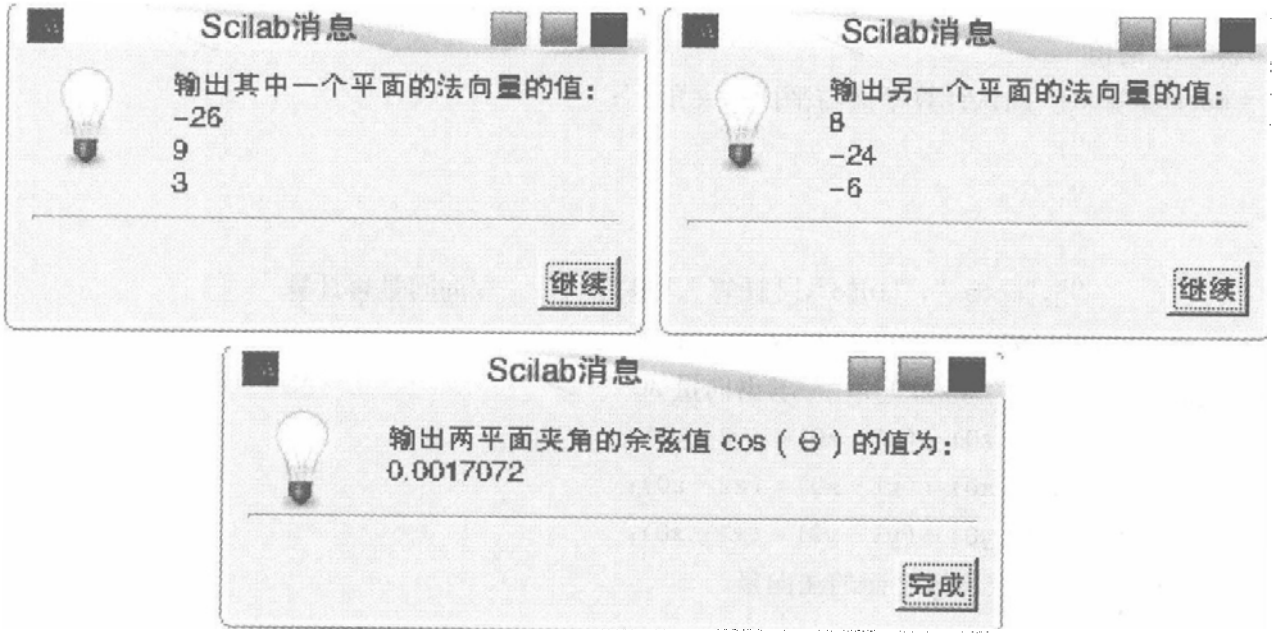


图 6.30 例(5)的执行结果

(6) 计算四面体的体积。

◆ **建模**

该算法主要用于在已知四面体的四个顶点的坐标，计算该四面体的体积。其中 $(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$ 为四面体 A-BCD 上四个点的坐标。

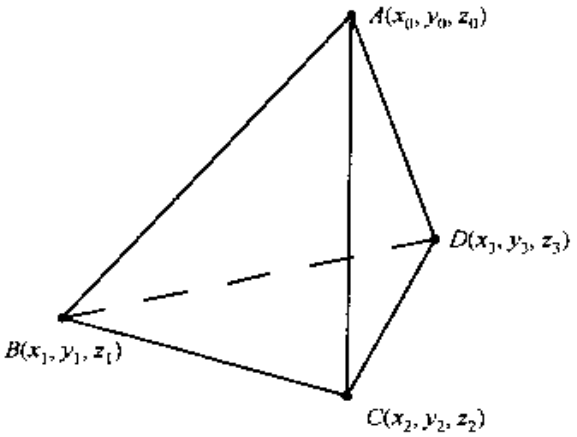


图 6.31　空间四面体

◆ **流程图**

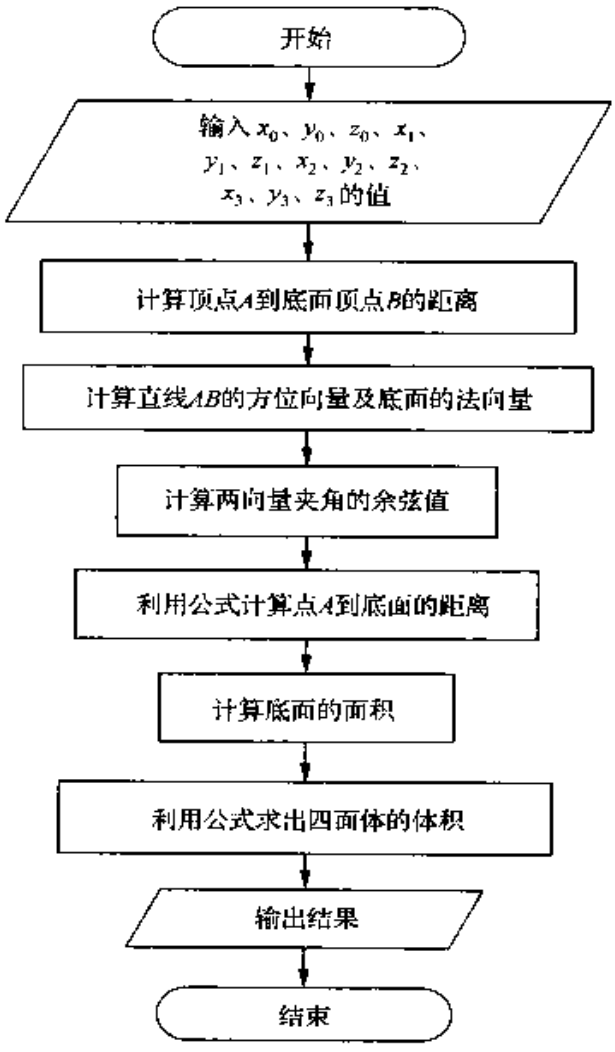


图 6.32　例(6)的流程图

◆ **Scilab 程序**

```
label1 = ['x0:';'y0:';'z0';'x1 :';'y1 :';'z1:';'x2 :';'y2:';'z2:';'x3:';'y3:';'z3:';];
B = x_mdialog(['请输入各点的坐标 :'], label1, ['0';'0';'1';'3';'6';'9';'3';'7';'6';'3';
'8';'1';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
z0 = evstr(B(3));
x1 = evstr(B(4));
y1 = evstr(B(5));
z1 = evstr(B(6));
x2 = evstr(B(7));
y2 = evstr(B(8));
z2 = evstr(B(9));
x3 = evstr(B(10));
y3 = evstr(B(11));
z3 = evstr(B(12));   //利用对话框来输入 x0、y0、z0、x1、y1、z1、x2、y2、z2、x3、y3、z3 的值
D = messagebox(['确认计算四面体的体积?';
                '';
                '';
                '';
                ''],"modal", "info",['计算','退出']);  //询问是否计算
d = ((x0 - x1)^2 + (y0 - y1)^2 + (z0 - z1)^2)^(1/2);  //计算 AB 两点的距离
  a = x0 - x1;
  b = y0 - y1;
  c = z0 - z1;
  n = [a,b,c]  //计算直线 AB 的方位向量
  a1 = ((y3 - y2) * (z2 - z1) - (y2 - y1) * (z3 - z2));
  b1 = ((z3 - z2) * (x2 - x1) - (x3 - x2) * (z2 - z1));
  c1 = ((x3 - x2) * (y2 - y1) - (x2 - x1) * (y3 - y2));
  m = [a1,b1,c1];        //计算底面的法向量

costh = abs((a * a1 + b * b1 + c * c1)/((a^2 + b^2 + c^2)^(1/2) * (a1^2 + b1^2 + c1^2)^(1/2)))
  //计算两向量夹角的余弦值
    h = d * costh;  //利用公式计算点 A 到底面的距离
    d1 = ((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2)^(1/2);
    d2 = ((x3 - x1)^2 + (y3 - y1)^2 + (z3 - z1)^2)^(1/2);
    d3 = ((x2 - x3)^2 + (y2 - y3)^2 + (z2 - z3)^2)^(1/2);   //计算底面三边的长度
    p = (d1 + d2 + d3)/2;
```

```
    s = (p * (p - d1) * (p - d2) * (p - d3))^(1/2);    //利用公式计算底面的面积
    V = 1/3 * s * h;    //利用公式计算四面体的体积
m = string(V);
E = messagebox['输出四面体的体积 V 的值:';m;],"modal", "info",['完成'])    //将体积的值符号化并输出
```

◆ 程序运行结果

输入 $x_0 = 0, y_0 = 0, z_0 = 1, x_1 = 3, y_1 = 6, z_1 = 9, x_2 = 3, y_2 = 7, z_2 = 6, x_3 = 3, y_3 = 8, z_3 = 1$ 时的结果：

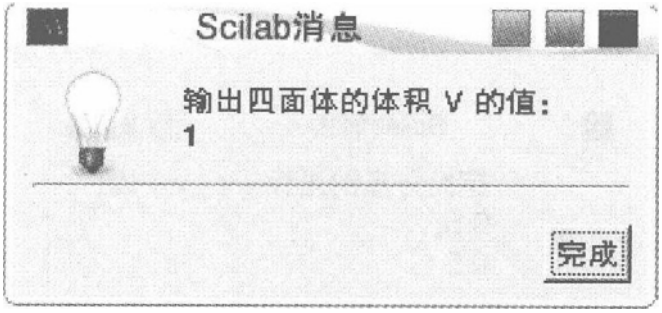


图 6.33　例(6)的执行结果

6.6　平面几何中的计算

(1) 已知两点的坐标 (x_1, y_1)、(x_2, y_2) 和定比系数 L，求定比分点的坐标。

◆ 建模

该算法主要用于计算已知两点的坐标 (x_1, y_1)、(x_2, y_2) 和定比系数 L，求定比分点的坐标。

◆ 流程图

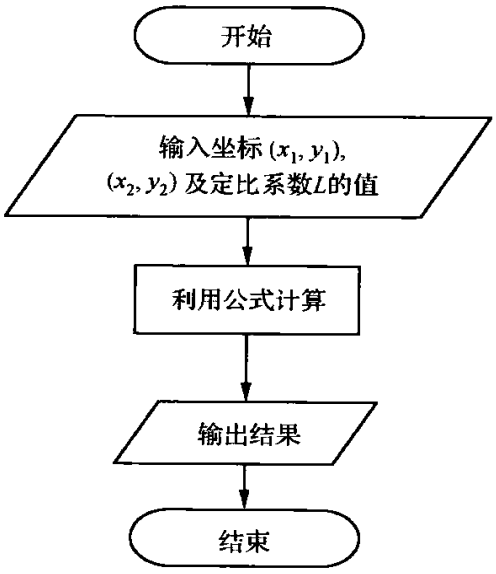


图 6.34　例(1)的流程图

◆ **Scilab 程序**

```
x1 = 1;y1 = 1;x2 = 0;y2 = 0;L = 1;  //输入两点坐标(x1,y1)、(x2,y2)
                                    及定比系数 L 的值
x = (x1 + L * x2)/(1 + L);          //利用公式计算出分点的横坐标
y = (y1 + L * y2)/(1 + L);          //利用公式计算出分点的纵坐标
m = string(x);     //将 x 的值符号化
n = string(y);     //将 y 的值符号化
E = messagebox(['定比分点的坐标(x,y)为:';m;n],"modal", "info",["完成"])  //利用对话框输出结果
```

◆ **程序运行结果**

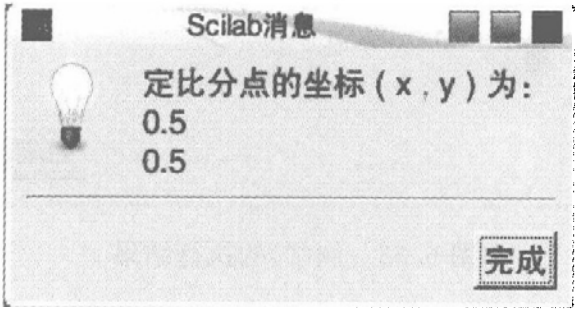


图 6.35 例(1)的执行结果

(2) 计算两点 (x_1,y_1)、(x_2,y_2) 之间的距离并作图表示。

◆ **建模**

该算法主要用于计算两点 (x_1,y_1)、(x_2,y_2) 之间的距离并作图表示。

◆ **流程图**

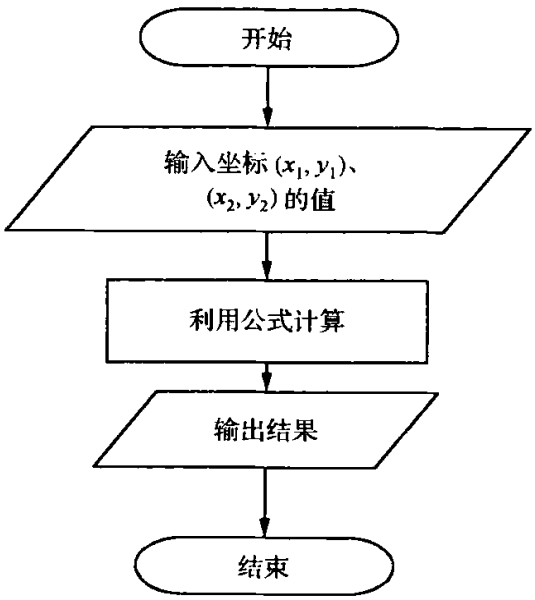


图 6.36 例(2)的流程图

◆ **Scilab 程序**

```
x1 = 0;y1 = 0;x2 = 1;y2 = 3;    //输入两点(x1,y1)、(x2,y2)的值
x = x1 : 0.01 : x2;        //给定 x 数组,确定范围及取点密度
d = sqrt((x2 - x1)^2 + (y2 - y1)^2); //利用公式计算这两点的距离
m = string(d);  //将 d 符号化
E = messagebox['输出距离 d  的值:';m;],"modal", "info",["退出" "几何表示"])
//利用对话框输出距离 d 的值
k = (y2 - y1)/(x2 - x1);  //计算这两点所在直线的距离
b = y1 - k * x1;   //计算这两点所在直线在 y 轴上的截距
plot(x,y = k * x + b,axesflag = 5);  //绘制这两点所在的直线
xset('color',21);  //设置坐标轴的颜色
xset('font size',2);  //设置坐标轴的单位长度
plot(y = y1,axesflag = 5,'b');  //绘制直线 y = y1
xset('color',21);//设置坐标轴的颜色
xset('font size',2);//设置坐标轴的单位长度
plot(x = x2,axesflag = 5,'b');//绘制直线 x = x2
xset('color',21);//设置坐标轴的颜色
xset('font size',2);//设置坐标轴的单位长度
d = sqrt((x2 - x1)^2 + (y2 - y1)^2);//利用公式计算这两点的距离
m = string(d);//将 d 符号化
E = messagebox['输出距离 h  的值:';m;],"modal", "info",["完成"]) //利用对话框输出距离 d 的值
```

◆ **程序运行结果**

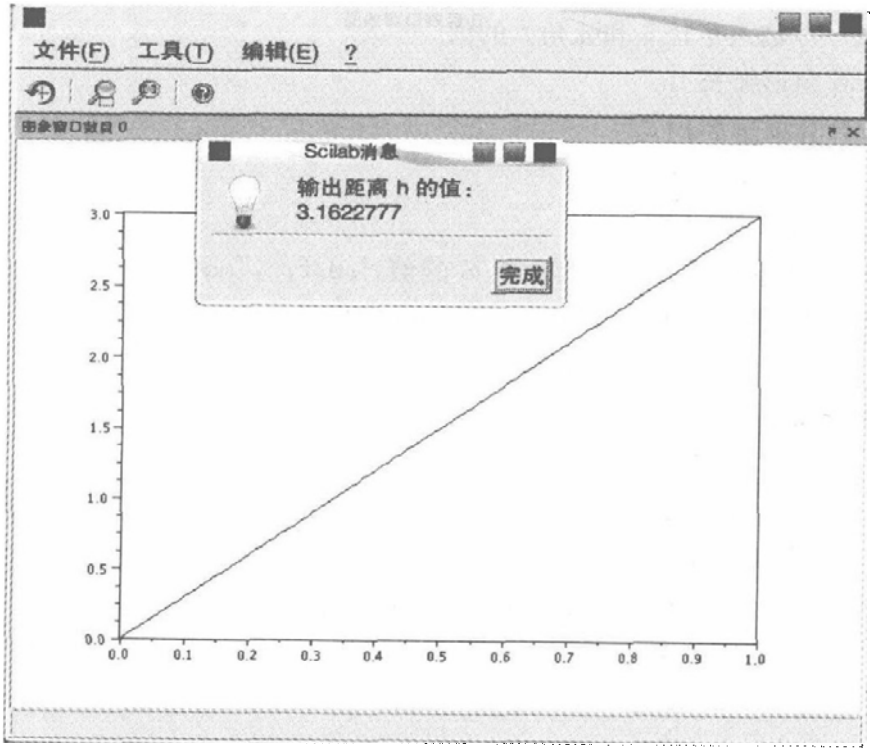


图 6.37　例(2)的执行结果

(3) 计算扇形的弧长及面积,并作图表示。

◆ **建模**

该算法主要用于在已知扇形的半径及弧角的情况下,计算扇形的弧长及面积,并作图表示。

◆ **流程图**

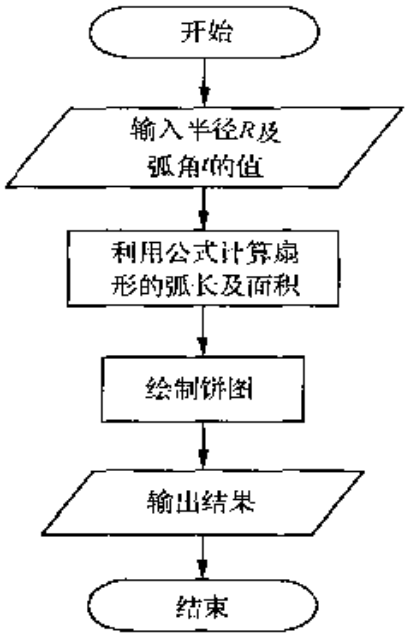


图 6.38 例(3)的流程图

◆ **Scilab 程序**

```
R=4;t= %pi/3;  //输入半径 R 和弧角 t 的值
L=t*R;  //计算扇形弧长 L
S=L*R/2;  //计算扇形面积 S
e=string(L);  //将 L 符号化
f=string(S);  //将 S 符号化
E=messagebox['输出扇形弧长 L 与其面积 S 的值:';e;f;],"modal", "info",["退出" "几何表示"]);  //利用对话框输出扇形弧长 L 与其面积 S 的值
pie([t 2* %pi-t],[1 1]);  //绘制饼图
L=t*R;  //计算扇形弧长 L
S=L*R/2;  //计算扇形面积 S
e=string(L);  //将 L 符号化
f=string(S);  //将 S 符号化
E=messagebox['输出扇形弧长 L 与其面积 S 的值:';e;f;],
"modal", "info",["完成"]);  //利用对话框输出扇形弧长 L 与其面积 S 的值
```

◆ **程序运行结果**

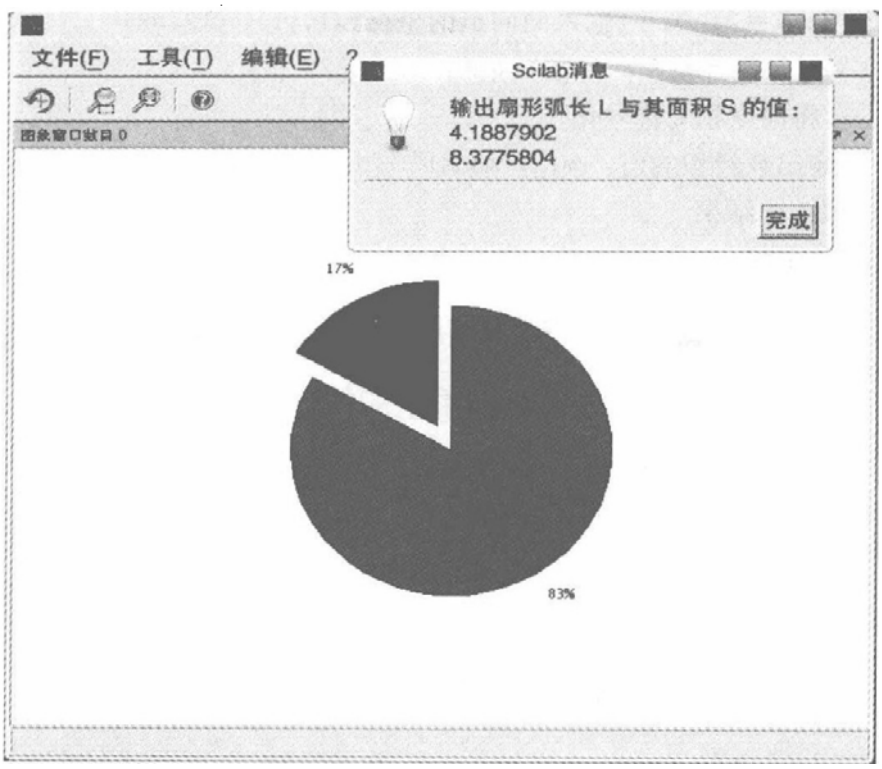


图 6.39　例(3)的执行结果

(4) 已知两个向量的坐标表示，计算他们的数量积。

◆ **建模**

该算法主要用于计算已知两向量的坐标 (x_1, y_1)、(x_2, y_2)，求两向量的数量积。

◆ **流程图**

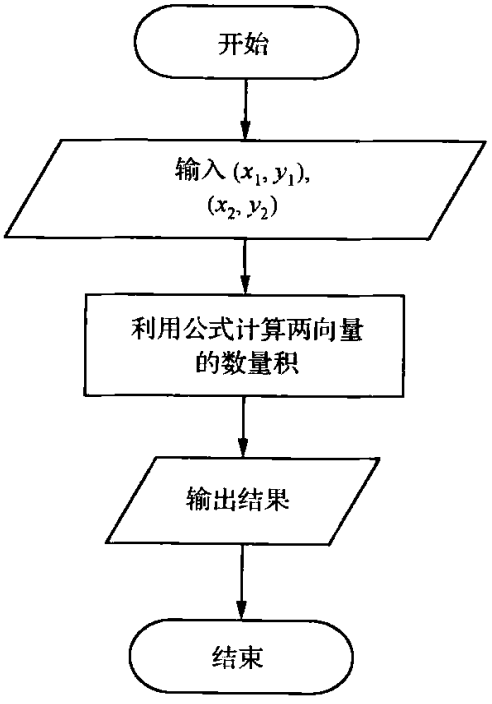


图 6.40　例(4)的流程图

◆ **Scilab 程序**

```
x1 = 4;y1 = 1;x2 = 1;y2 = 2;      //输入两向量的坐标(x1,y1)、(x2,y2)
s = x1 * x2 + y1 * y2;    //计算两向量的数量积
m = string(s);    //将向量积 s 符号化
E = messagebox(['输出数量积的值:';m;],"modal", "info",["完成"]);
    //利用对话框输出结果
```

◆ **程序运行结果**

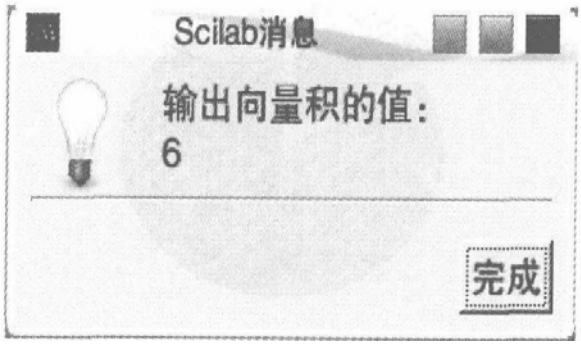


图 6.41　例(4)的执行结果

(5) 已知两点的坐标 (x_1,y_1)、(x_2,y_2)，求过这两点的直线的斜率并作图表示。

◆ **建模**

该算法主要用于计算已知两点的坐标 (x_1,y_1)、(x_2,y_2)，求过这两点的直线的斜率并作图表示。

◆ **流程图**

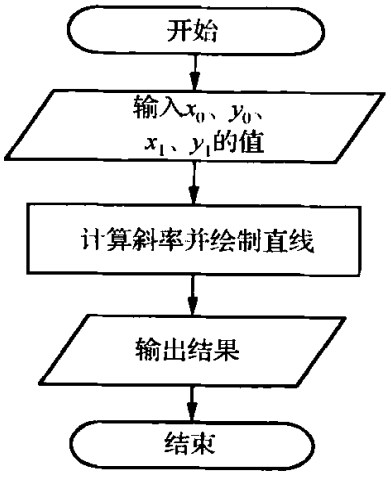


图 6.42　例(5)的流程图

◆ **Scilab 程序**

```
x0 = 0;y0 = 0;x1 = 1;y1 = 2;      //输入两点(x1,y1)、(x2,y2)的坐标
k = (y1 - y0)/(x1 - x0);      //计算斜率
m = string(k);      //将 k 的值符号化
```

```
E = messagebox(['输出斜率 k 的值:';m;],"modal", "info",
["退出" "几何表示"])  //利用对话框输出斜率的值并询问是否作图
x = -10 : 0.01 : 10;      //给定 x 数组,确定范围及取点密度
k = (y1 - y0)/(x1 - x0);  //计算斜率
y = k * x + y1 - k * x1;  //计算出每个 x 所对应的 y 值
plot2d(x,y,axesflag = 5);  //绘制图像
xset('color',21);         //设置坐标轴的颜色
xset('font size',2);      //设置坐标轴的单位长度
xtitle('y = k * x + y1 - k * x1');  //在坐标轴上显示函数表达式
k = (y1 - y0)/(x1 - x0);  //计算斜率
m = string(k);    //将 k 的值符号化
E = messagebox(['输出斜率 k 的值:';m;],"modal", "info",["完成"])
          //利用对话框输出斜率的值
```

◆ **程序运行结果**

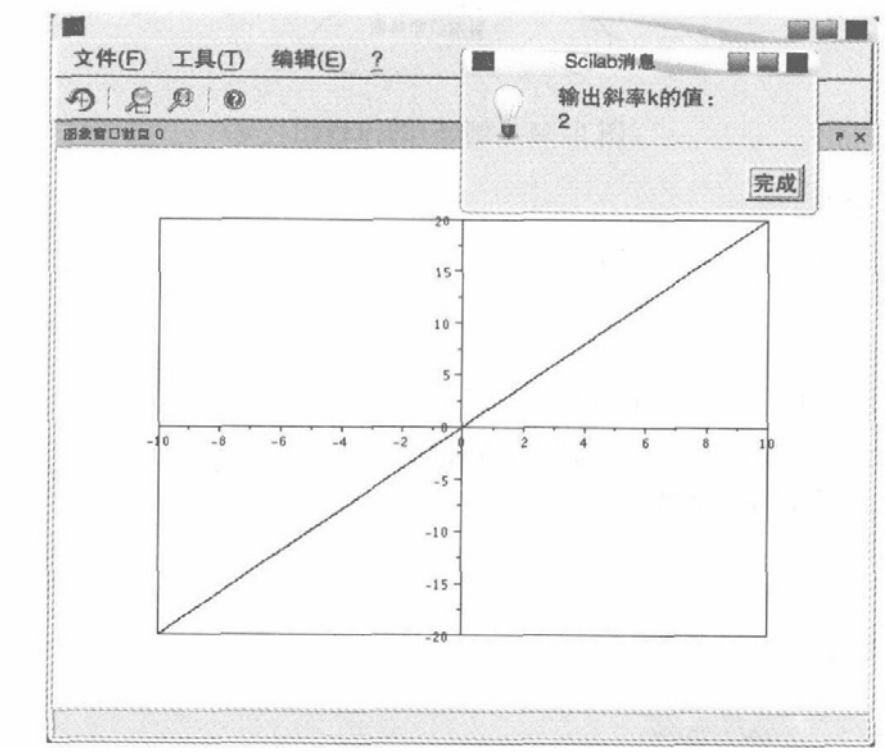


图 6.43　例(5)的执行结果

(6) 计算长方形的面积。

◆ **建模**

已知长方形的长和宽,利用公式进行求解。

◆ 流程图

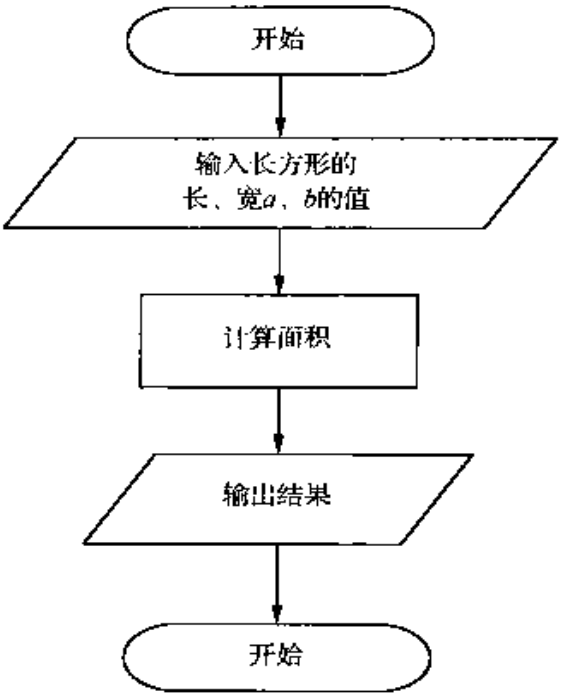


图 6.44 例(6)的流程图

◆ Scilab 程序

```
function changfangxing()      //定义函数名
clc
clear     //清除内存
into = messagebox['欢迎使用 Scilab! ^_^';
'程序：面积的计算';'描述:';'计算长方形的面积';],
['确定']);     //程序功能介绍
label1 = ['a :';'b :';];
B = x_mdialog['请输入 a 和 b 的值 :'],...
  label1,['8';'7';]);
a = evstr(B(1));
b = evstr(B(2));     //输入设置
if B = = []
quit
end          //若选择取消按钮
输入参数
D = messagebox['确认要进行长方形面积的计算?'],
['退出','计算'];
//编程
  if a>0&b>0
```

```
  s = a * b;          //判断并计算
  c = string(s);      //将 s 符号化
  E = messagebox(['输出长方形的面积的值:';c;],['完成'])
  else
    printf('a<0 or b<0')
    E = messagebox(['a<0 or b<0  !!! ';],['完成'])      //若 a<0 或 b<0 则推出
  end
endfunction
```

◆ **程序运行结果**

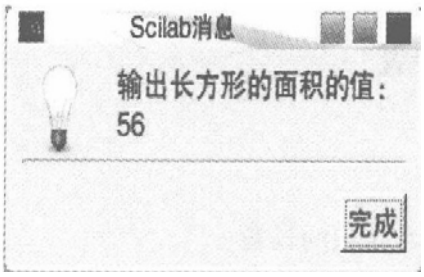


图 6.45　例(6)的执行结果

(7) 已知菱形的对角线计算菱形的面积。

◆ **建模**

在已知菱形对角线 a、b 长度的情况下，可利用公式 $s=\frac{1}{2}ab$ 来求菱形的面积。

◆ **流程图**

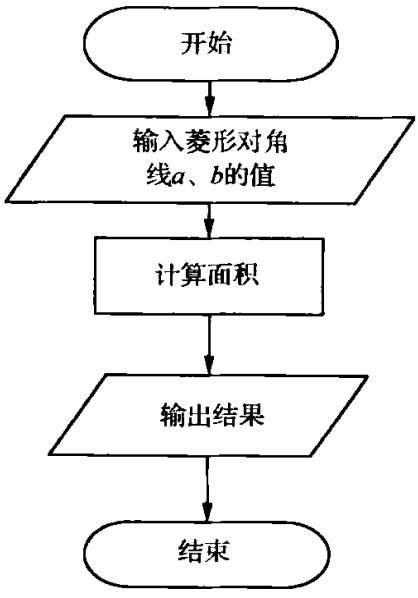


图 6.46　例(7)的流程图

◆ **Scilab 程序**

```
function lingxing()      //定义函数名
clc
clear                    //清除内存
into = messagebox(['欢迎使用 Scilab! ^_^';
'程序：面积的计算';'描述:';'计算菱形的面积
(已知对角线)'],['确定']);  //程序功能介绍
label1 = ['a :';'b :';];
B = x_mdialog(['请输入 a 和 b 的值 ：'],...
        label1,['8';'7';]);
a = evstr(B(1));
b = evstr(B(2));         //输入设置
if B = = []
quit
end          //若选择取消按钮
D = messagebox(['确认要进行菱形面积的计算？'],
['退出','计算']);
//编程
  if a>0&b>0
     S = a * b/2;        //判断 a、b 的值并利用公式计算
     c = string(S);  //将 S 的值符号化
  E = messagebox(['输出菱形的面积的值:';c;],['完成'])
  else
     printf('a<0 or b<0')
     E = messagebox(['a<0 or b<0  !!! ';],['完成'])
  end
endfunction
```

◆ **程序运行结果**

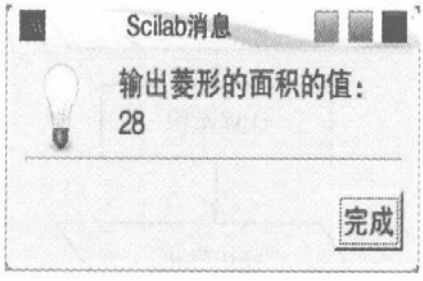


图 6.47　例(7)的执行结果

(8) 已知三角形的三边长 a、b、c，计算三角形的面积和三边上的高。

◆ **建模**

该算法主要用于已知三角形的三边长 a、b、c，计算三角形的面积和三边上的高。

◆ **流程图**

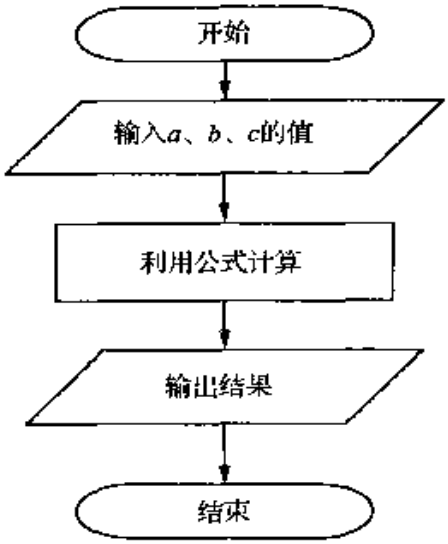


图 6.48 例(8)的流程图

◆ **Scilab 程序**

```
a=3;b=4;c=5;      //输入 a,b,c 的值
s=(a+b+c)/2;

area=sqrt(s*(s-a)*(s-b)*(s-c));//利用公式计算出
                                该三角形的面积
ha=2*area/a;                   //计算出边 a 上的高

hb=2*area/b;                   //计算出边 b 上的高

hc=2*area/c;                   //计算出边 c 上的高
d=string(area);                //将 area 符号化
e=string(ha);                  //将 ha 符号化
g=string(hb);                  //将 hb 符号化
h=string(hc);                  //将 hc 符号化

E=messagebox(['输出三角形的面积 S 及边 a 上的高、边 b 上的高、边 c 上的高:';d;e;g;h;],
"modal","info",["完成"])  //用对话框输出结果
```

◆ **程序运行结果**(图 6.49)

(9) 已知圆的半径 R，计算圆的面积。

◆ **建模**

该算法主要用于判断并计算圆的面积。

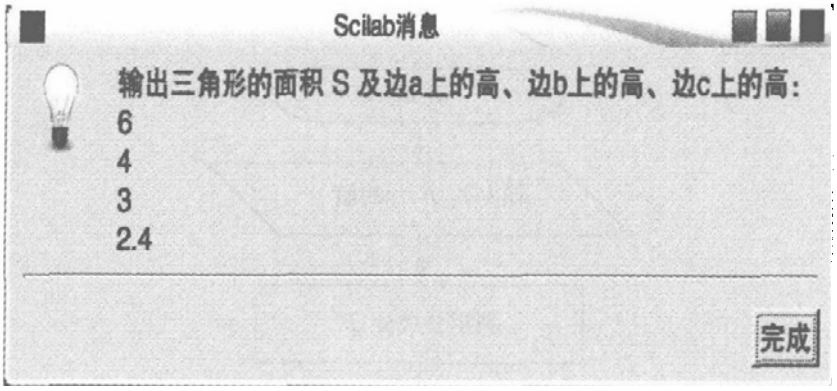


图 6.49 例(8)的执行结果

◆ **流程图**

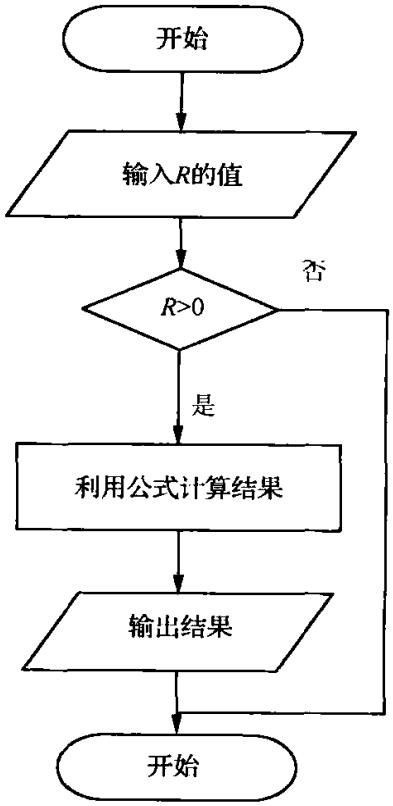


图 6.50 例(9)的流程图

◆ **Scilab 程序**

```
R=5;                    //输入 R 的值
if R>0                  //利用判断语句来判断当 R>0 的时候就计算出该圆的面积
S= %pi*(R^2);                    //利用公式计算出圆的面积

disp( R,'R=',S,'S=');    //输出半径和面积
  c=string(S);           //将 S 符号化
E=messagebox(['输出圆的面积的值:';c;],"modal", "info",["完成"])
                //利用对话框输出圆的面积
```

```
else

  printf('R<0');
E = messagebox(['R 小于或等于 0 啦  !!! ';],"modal", "info",["完成"])  //若 R≤0 就输出
提示
end
```

◆ **程序运行结果**

当 $R = 5$ 时：

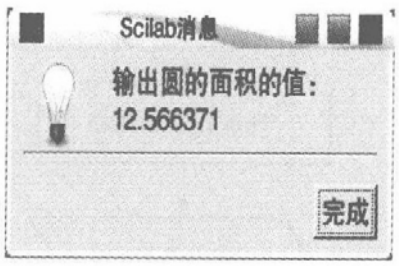


图 6.51 例(9)的执行结果

6.7 平面几何中的作图

(1) 已知 k_1 和 b_1，绘制反比例函数 $y = \frac{k_1}{x} + b_1$ 的图像。

◆ **建模**

该算法主要用于已知 k_1 和 b_1，绘制反比例函数 $y = \frac{k_1}{x} + b_1$ 的图像。

◆ **流程图**(图 6.52)

◆ **Scilab 程序**

```
k1 = 1;b1 = 0;m = -10;n = 10;  //输入 k1、b1 及区间左右端点 m、n 的值
if m<0&n<0
    x1 = m : 0.01 : n;      //m,n 同时小于 0 时,给定 x1 数组,确定范围及取点密度;
    y = k1./x1 + b1;        //计算出每个 x1 所对应的 y 值
    e = k1/n + b1;          //计算出右端点所对应的 y 值
    plot2d(x1,y,axesflag = 5,rect = [-abs(m) - 1,e,abs(m) + 1,abs(e)]);  //绘图
    xset('color',21);       //设置坐标轴的颜色
    xset('font size',2);    //设置坐标轴的单位长度
    end
if m>0&n>0
    x1 = m : 0.01 : n;      //m,n 同时大于 0 时,给定 x1 数组,确定范围及取点密度;
    y = k1./x1 + b1;        //计算出每个 x1 所对应的 y 值
```

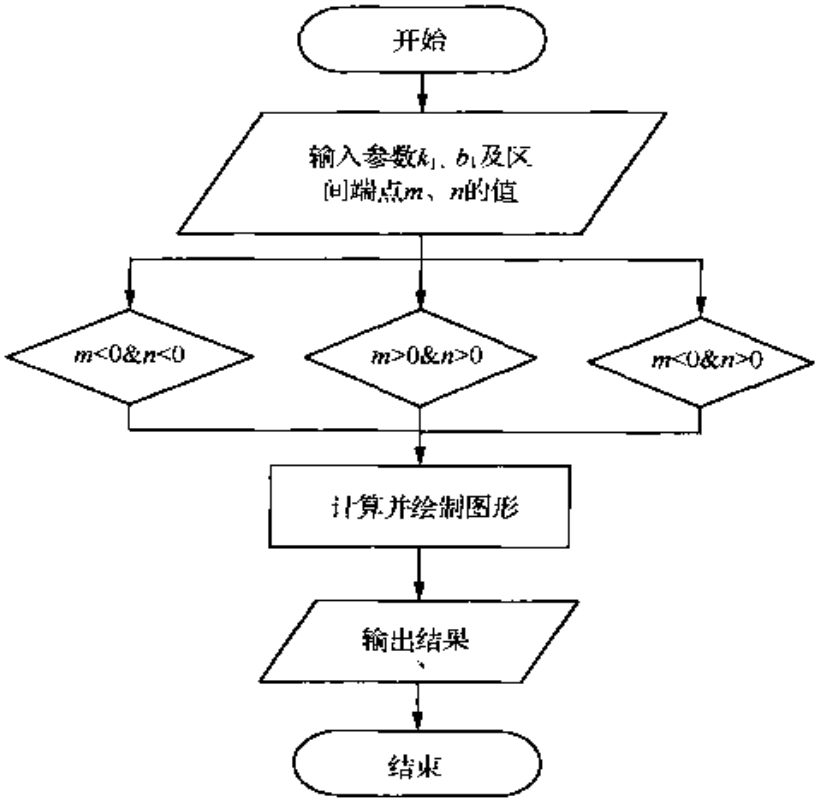


图 6.52 例(1)的流程图

```
    f = k1/m + b1;        //计算出左端点所对应的 y 值
  plot2d(x1,y,axesflag = 5,rect = [ - abs(m) - 1, - abs(f),abs(m) + 1,abs(f)]);    //绘图
    xset('color',21);  //设置坐标轴的颜色
    xset('font size',2);   //设置坐标轴的单位长度
  end
   if m<0&n>0
    t = m : 0.0001 : - 0.01;//m<0,n>0 时,给定 t 数组,确定范围及取点密度;
    y3 = k1./t + b1;        //计算出每个 t 所对应的 y 值
    b = k1/( - 0.3) + b1;    //计算出接近 0⁻ 的某个函数值
    d = k1/(0.3) + b1;       //计算出接近 0⁺ 的某个函数值
    l = 0.01 : 0.0001 : n;  //给定 l 数组,确定范围及取点密度
    y4 = k1./l + b1;        //计算出每个 l 所对应的 y 值
    plot2d(t,y3,axesflag = 5,rect = [ - abs(m) - 8,b,abs(n) + 8,d]);//绘制函数的左半条图像
    xset('color',21);  //设置坐标轴的颜色
    xset('font size',2);  //设置坐标轴的单位长度
    plot2d(l,y4,axesflag = 5,rect = [ - abs(m) - 8,b,abs(n) + 8,d]);//绘制函数的右半条图像
    xset('color',21);//设置坐标轴的颜色
```

```
    xset('font size',2);  //设置坐标轴的单位长度
end
```

◆ **程序运行结果**

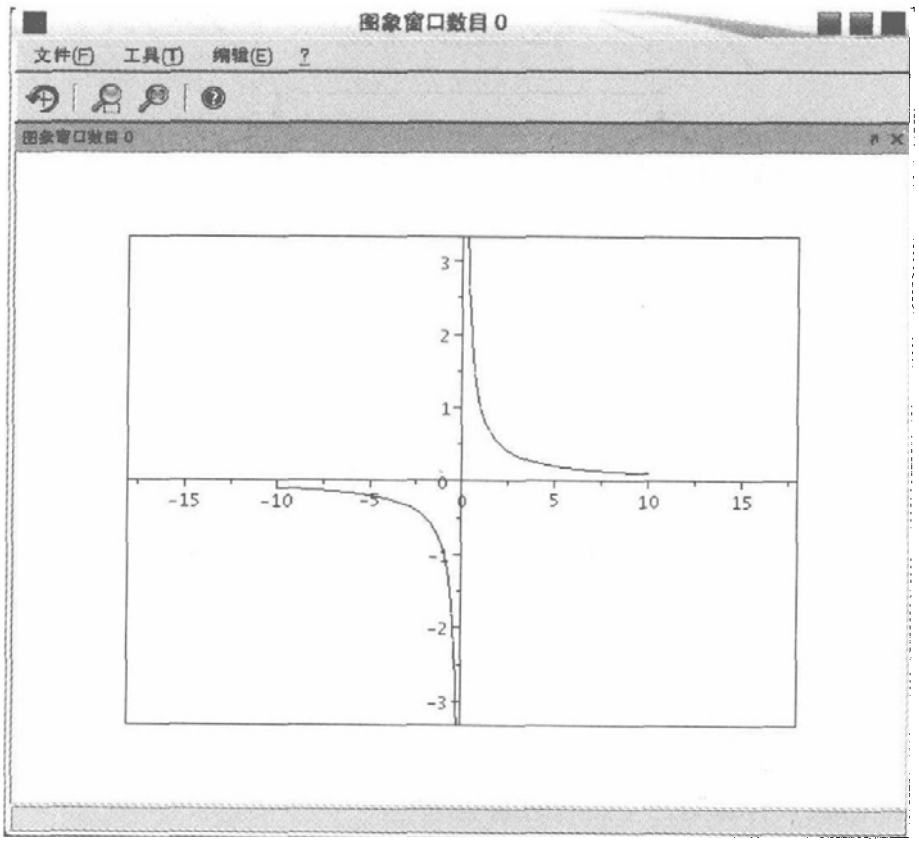


图 6.53　例(1)的执行结果

(2) 已知反比例函数 $y=\frac{k_1}{x}+b_1$ 中的系数和常数项 k_1、b_1 以及一次函数 $y=k_2x+b_2$ 的系数和常数项 k_2、b_2，计算一元一次函数与反比例函数的焦点并绘制图形。

◆ **建模**

该算法主要用计算一元一次函数与反比例函数的焦点并绘制图形。

◆ **流程图**(图 6.54)

◆ **Scilab 程序**

```
k1 = 1;b1 = 0;k2 = 1;b2 = 0;m = - 10;n = 10; //请输入 k1,b1,k2,b2,m,n 的值,k1、b1 表示反比例函数中的系数和常数项,k2、b2 为一次函数的系数和常数项,m、n 为它的一个区间
if m * n>0
  x1 = m : 0.01 : n;
  x2 = m : 0.01 : n;//当 m,n 同号时给定 x1 及 x2 数组,确定范围及取点密度
  z = [x1;x2]';    //将 x1 及 x2 数组转置
  y = [k1./x1 + b1;k2. * x2 + b2]'; //计算反比例函数及一次函数所对应的 y 值,并将其转置
   r = (b2 - b1)^2 + 4 * k1 * k2;
```

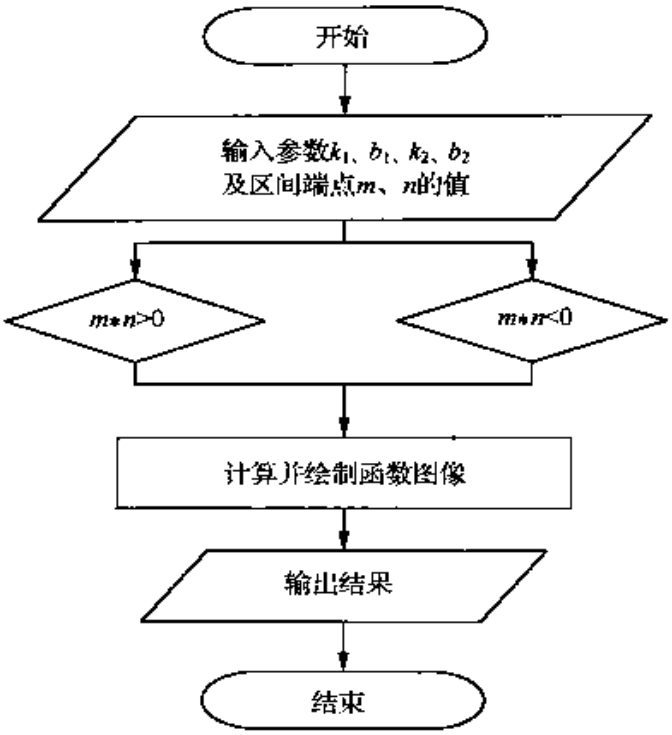


图 6.54 例(2)的流程图

```
x1 = ((b1 - b2) - r ^(1/2))/(2 * k1);//计算其中一个焦点的横坐标
x2 = ((b1 - b2) + r ^(1/2))/(2 * k1);//计算另一个焦点的横坐标
y1 = k2 * x1 + b2;//计算第一个焦点的纵坐标
y2 = k2 * x2 + b2;//计算第二个焦点的纵坐标
d = string(x1);
e = string(x2);
g = string(y1);
h = string(y2);   //将 x1,x2,y1,y2 的值符号化
E = messagebox(['输出其中一个焦点的坐标(x1,y1)和另一个焦点的坐标(x2,y2)的值为:
';d;g;e;h;],"modal", "info",["绘图"  "退出" ])  //利用对话框输出两个焦点的坐标
plot2d(z,y,axesflag = 5,rect = [x1 - 8,y1 - 8,x2 + 8,y2 + 8]); //绘制此时的函数图像
xset('color',21);     //设置坐标轴的颜色
xset('font size',2); //设置坐标轴的单位长度
else  //当 m,n 异号的情况
r = (b2 - b1)^2 + 4 * k1 * k2;
x1 = ((b1 - b2) - r ^(1/2))/(2 * k1);//计算其中一个焦点的横坐标
x2 = ((b1 - b2) + r ^(1/2))/(2 * k1);//计算另一个焦点的横坐标
y1 = k2 * x1 + b2;                    //计算第一个焦点的纵坐标
y2 = k2 * x2 + b2;                  //计算第二个焦点的纵坐标
d = string(x1);
e = string(x2);
g = string(y1);
```

```
    h = string(y2);    //将 x1,x2,y1,y2 的值符号化
  E = messagebox(['输出其中一个焦点的坐标(x1,y1)和另一个焦点的坐标(x2,y2)的值为:';
d;g;e;h;],"modal", "info",["绘图"  "退出" ])    //利用对话框输出两个焦点的坐标
    t = m : 0.0001 : -0.01;    //给定 t 数组,确定范围及取点密度
    y3 = [k1./t + b1;k2. * t + b2]';  //计算反比例函数及一次函数所对应的 y 值,并将其
转置
    l = 0.01 : 0.0001 : n;    //给定 t 数组,确定范围及取点密度
    y4 = [k1./l + b1;k2. * l + b2]';  //计算反比例函数及一次函数所对应的 y 值,并将其
转置
    plot2d(t,y3,axesflag = 5,rect = [-abs(x1) - 8, -abs(y1) - 8,abs(x2) + 8,abs(y2) + 8]);
                          //绘制数组 t 对应的函数图像
    xset('color',21);    //设置坐标轴的颜色
    xset('font size',2); //设置坐标轴的单位长度
    plot2d(l,y4,axesflag = 5,rect = [-abs(x1) - 8, -abs(y1) - 8,abs(x2) + 8,abs(y2) + 8]);
                      //绘制数组 l 对应的函数图像
    xset('color',21);    //设置坐标轴的颜色
    xset('font size',2); //设置坐标轴的单位长度
    e1 = string(k1);
    h1 = string(b1);
    g1 = string(k2);
    t1 = string(b2);
    n1 = string(x1);
    m1 = string(y1);
    n2 = string(x2);
m2 = string(y2);    //将 k1,b1,k2,b2,x1,y1,x2,y2 符号化
xtitle('y = ' + e1 + '/x + ' + h1 + '','y = ' + g1 + 'x + ' + t1 + '');  //在坐标轴上方输出反比例
函数及一次函数的函数表达式
  end
    r = (b2 - b1)^2 + 4 * k1 * k2;
    x1 = ((b1 - b2) - r^(1/2))/(2 * k1);  //计算其中一个焦点的横坐标
    x2 = ((b1 - b2) + r^(1/2))/(2 * k1);  //计算另一个焦点的横坐标
    y1 = k2 * x1 + b2;  //计算第一个焦点的纵坐标
    y2 = k2 * x2 + b2;  //计算第二个焦点的纵坐标
    d = string(x1);
    e = string(x2);
    g = string(y1);
    h = string(y2);  //将 x1,y1,x2,y2 符号化
E = messagebox(['输出其中一个焦点的坐标(x1,y1)和另一个焦点的坐标(x2,y2)的值为:';d;
```

```
g;e;h;],"modal", "info",["完成"])  //利用对话框输出两个焦点的坐标
```

◆ **程序运行结果**

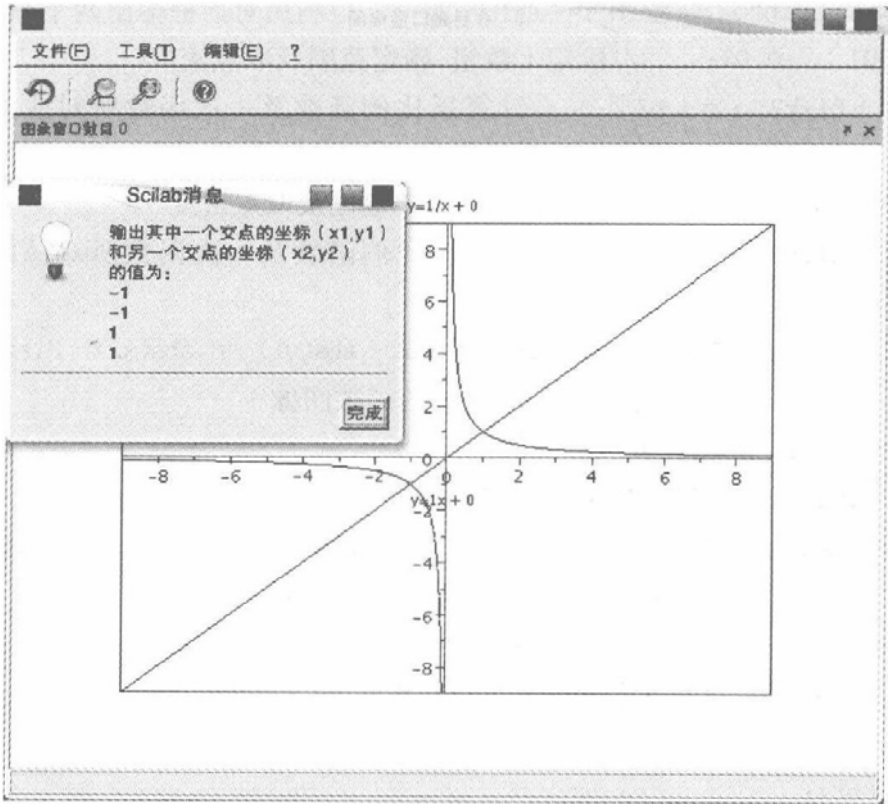


图 6.55　例(2)的执行结果

(3) 绘制分段函数的图像。

◆ **建模**

该算法主要用于已知每个区间段上的函数表达式，绘制分段函数的图像。

◆ **流程图**

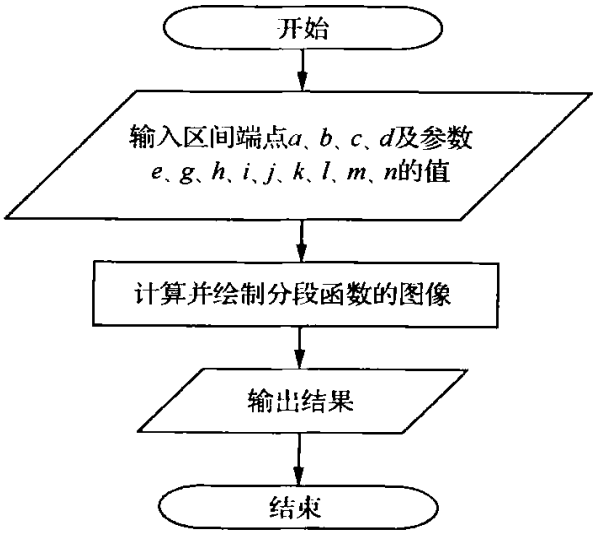


图 6.56　例(3)的流程图

◆ **Scilab 程序**

```
label1 = ['a:';'b:';'c:';'d :';'e :';'g:';'h :';'i:';'j:';'k:';'l:';'m:';'n';];
B = x_mdialog(['请输入 a,b,c,d,e,g,h,i,j,k,l,m,n 的值';'(a,b)为 y1 的区间,e,g,h 为 y1 = e * x^2 + g * x + h 的系数';'(b,c)为 y2 的区间,i,j,k 为 y2 = i * x ^2 + j * x + k 的系数';'(c,d)为 y3 的区间,l,m,n 为 y3 = l * x ^2 + m * x + n 的系数 :'],label1,['-10';'-6';'-1';'10';'0';'2';'3';'0';'6';'0';'0';'1';'9']);
a = evstr(B(1));
b = evstr(B(2));
c = evstr(B(3));
d = evstr(B(4));
e = evstr(B(5));
g = evstr(B(6));
h = evstr(B(7));
i = evstr(B(8));
j = evstr(B(9));
k = evstr(B(10));
l = evstr(B(11));
m = evstr(B(12));
n = evstr(B(13));      //以上的作用是利用对话框来输入 a,b,c,d,e,g,h,i,j,k,l,m,n 的值
D = messagebox['确认进行绘制分段函数的图像?';
                ";
                ";
                ";
                "],"modal", "info",["绘制"  "退出"]); //利用对话框来询问是否绘制图像
x1 = a : 0.0001 : b;      //给定(a,b)段上的数组 x1,确定范围及取点密度
y1 = e * x1^2 + g * x1 + h;  //计算出对应的 y 值
plot2d(x1,y1,axesflag = 5);  //绘制(a,b)段上的函数图像
x2 = b + 0.0001 : 0.0001 : c;  //给定(b,c)段上的数组 x2,确定范围及取点密度
y2 = i * x2^2 + j * x2 + k;   //计算出对应的 y 值
plot2d(x2,y2,axesflag = 5);  //绘制(b,c)段上的函数图像
x3 = c + 0.0001 : 0.0001 : d;   //给定(c,d)段上的数组 x3,确定范围及取点密度
y3 = l * x3^2 + m * x3 + n;   //计算出对应的 y 值
plot2d(x3,y3,axesflag = 5); //绘制(c,d)段上的函数图像
xset('color',21);  //设置坐标轴的颜色
xset('font size',2);  //设置坐标轴的单位长度
```

◆ **程序运行结果**

输入 $a=-10, b=-6, c=-1, d=10, e=0, g=2, h=3, i=0, j=6, k=0, l=0, m=1, n=9$ 后得到的结果：

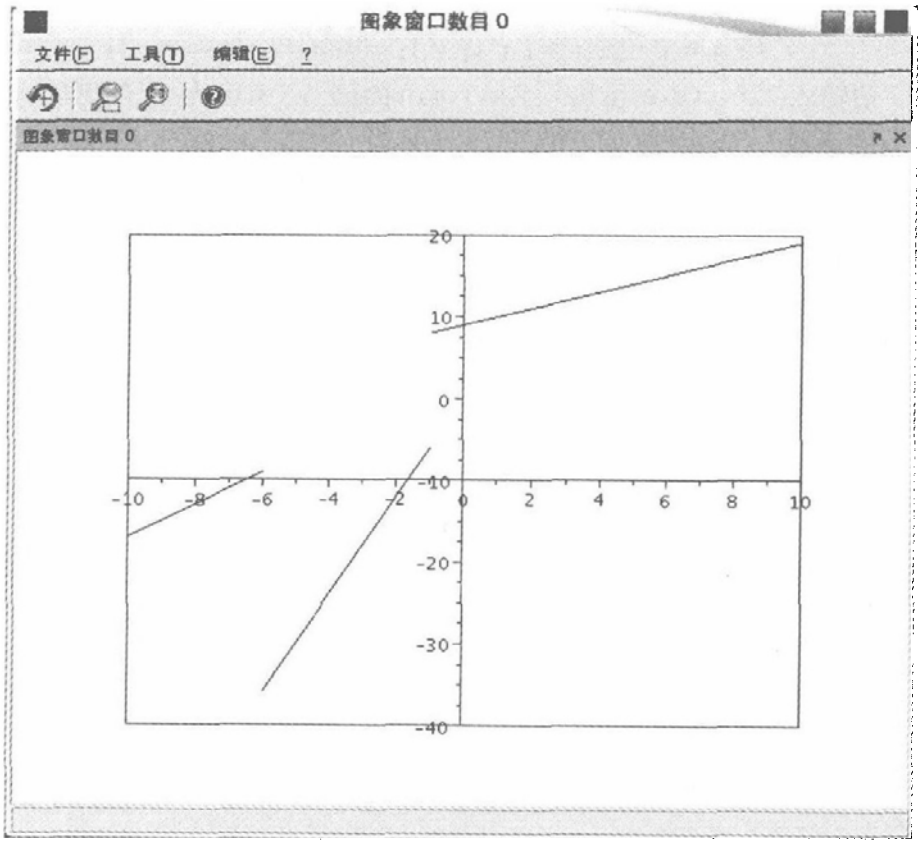


图 6.57 例(3)的执行结果

(4) 计算双曲线 $\frac{(x-e)^2}{a^2}-\frac{(y-g)^2}{b^2}=1$ 的离心率并作图。

◆ **建模**

该算法主要用于判断并计算双曲线 $\frac{(x-e)^2}{a^2}-\frac{(y-g)^2}{b^2}=1$ 的离心率并作图。

a、b、e、g 为 $\frac{(x-e)^2}{a^2}-\frac{(y-g)^2}{b^2}=1$ 中 a、b、e、g 的值，$k=0$ 表示焦点在 x 轴上，$k=1$ 表示焦点在 y 轴上。

◆ **流程图**(图 6.58)

◆ **Scilab 程序**

```
label1 = ['a:';'b:';'e';'g';'k';];
B = x_mdialog(['请输入 (x-e)^2/a^2—(y-g)^2/b^2=1 中的 a、b、e、g 、k 的值';' k=0 表示焦点在 x 轴上,k=1 表示焦点在 y 轴上 :'],label1,['4';'5';'0';'0';'0';]);
a = evstr(B(1));
b = evstr(B(2));
e = evstr(B(3));
```

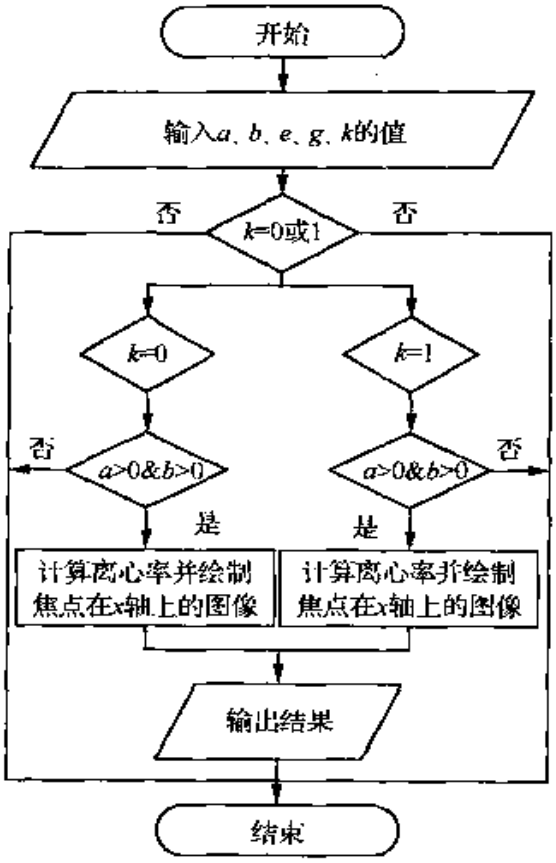


图 6.58 例(4)的流程图

```
g = evstr(B(4));
k = evstr(B(5));      //以上的作用是利用对话框来输入 a,b,e,g,k 的值
if  k = = 0     //判断焦点的位置 (焦点分别在 X 轴)
   t = 0 : 0.01 : 2 * %pi;      //给定数组 t,确定范围及取点密度
  if a>0&b>0          //若 a、b 均大于 0,则计算作图;否则输出错误信息
     x = a. /cos(t) + e;      //双曲线的参数方程
     y = b * tan(t) + g;
plot2d(x,y,frameflag = 3,rect = [ - (abs(a) + abs(e) + 9), - (abs(b) + abs(g) + 5),abs(a) +
abs(e) + 9,abs(b) + abs(g) + 5],axesflag = 5);      //绘制双曲线的图像
     xset('color',21);        //设置坐标轴的颜色
     xset('font size',2);      //设置坐标轴的单位长度
     a1 = string(a);
     b1 = string(b);
     e1 = string(e);
     g1 = string(g);      //将 a、b、e、g 的值符号化
     xtitle('(x - ' + e1 + ')^2/' + a1 + '^2 - (y - ' + g1 + ')^2/' + b1 + '^2 = 1');    //在坐标轴上
显示双曲线的方程
     c = (a^2 + b^2)^(1/2);
```

```
    E = c/a;        //计算离心率
    m = string(E);
    E = messagebox['输出离心率 e 的值:';m;],"modal", "info",['完成'])   //利用对话框
输出离心率的值
  else
E = messagebox['Error : a 、b  的值输入错误 !!!:';],"modal", "info",['完成'])
  end

elseif  k = = 1   //焦点分别在 Y 轴

  t = 0 : 0.01 : 2 * %pi;   //给定数组 t,确定范围及取点密度
  if a>0&b>0
    y = a /cos(t) + e;  //双曲线的参数方程
    x = b * tan(t) + g;
plot2d(x,y,frameflag = 3,rect = [ -(abs(b) + abs(e) + 9), -(abs(a) + abs(g) + 5),abs(b) +
abs(e) + 9,abs(a) + abs(g) + 5],axesflag = 5);     //绘制双曲线的图像
    xset('color',21);        //设置坐标轴的颜色
    xset('font size',2);     //设置坐标轴的单位长度
    a1 = string(b);
    b1 = string(a);
    e1 = string(e);
    g1 = string(g);     //将 a、b、e、g 符号化
    xtitle('(y - ' + e1 + ')^2/' + a1 + '^2 - (x - ' + g1 + ')^2/' + b1 + '^2 = 1');   //在坐标轴上
显示双曲线的方程
    c = (a^2 + b^2)^(1/2);
    E = c/a;      //计算离心率
   m = string(E);
    E = messagebox['输出离心率 e 的值:';m;],"modal", "info",['完成'])   //利用对话框
输出离心率的值
else
E = messagebox['Error : a 、b  的值输入错误 !!! ';],"modal", "info",['完成'])
end
end
```

◆ **程序运行结果**

当 $a = 4, b = 5, e = 0, g = 0, k = 0$ 时的结果：

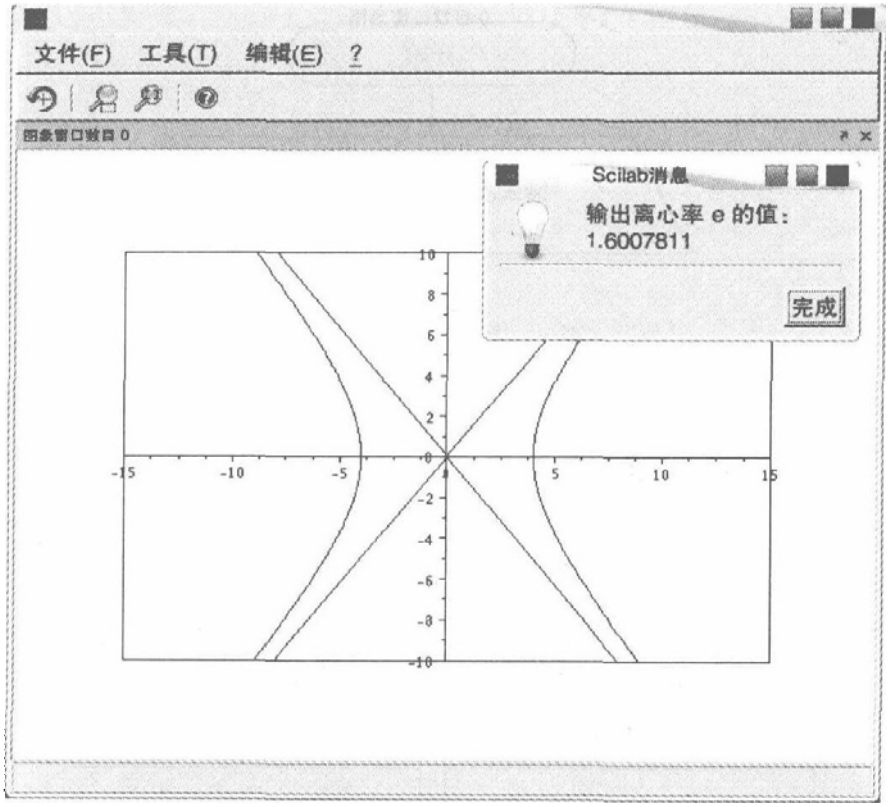


图 6.59　例(4)的执行结果

(5) 计算椭圆 $\frac{(x-e)^2}{a^2}+\frac{(y-g)^2}{b^2}=1$ 的离心率并作图。

◆ **建模**

该算法主要用于判断并计算椭圆 $\frac{(x-e)^2}{a^2}+\frac{(y-g)^2}{b^2}=1$ 的离心率并作图。a、b、e、g 为 $\frac{(x-e)^2}{a^2}+\frac{(y-g)^2}{b^2}=1$ 中 a、b、e、g 的值，$k=0$ 表示焦点在 x 轴上，$k=1$ 表示焦点在 y 轴上。

◆ **流程图**(图 6.60)

◆ **Scilab 程序**

```
label1 = ['a :';'b :';'e :';'g :';'k :';];
B = x_mdialog(['请输入 a ,b ,e ,g 的值,a ,  b , e , g 分别为 (x - e)^2/a^2 + (y - g)^2/b^2 = 1
或 (y - e)^2/a^2 + (x - g)^2/b^2 = 1 中的数 ';'k = 0 表示焦点在 x 轴上,k = 1 表示焦点在 y 轴上
  : '],... label1,['8';'5';'0';'0';'1';]);
a = evstr(B(1));
b = evstr(B(2));
e = evstr(B(3));
g = evstr(B(4));
k = evstr(B(5));        //以上的作用是利用对话框来输入 a,b,e,g,k 的值
  if k = = 0      //焦点分别在 X 轴
```

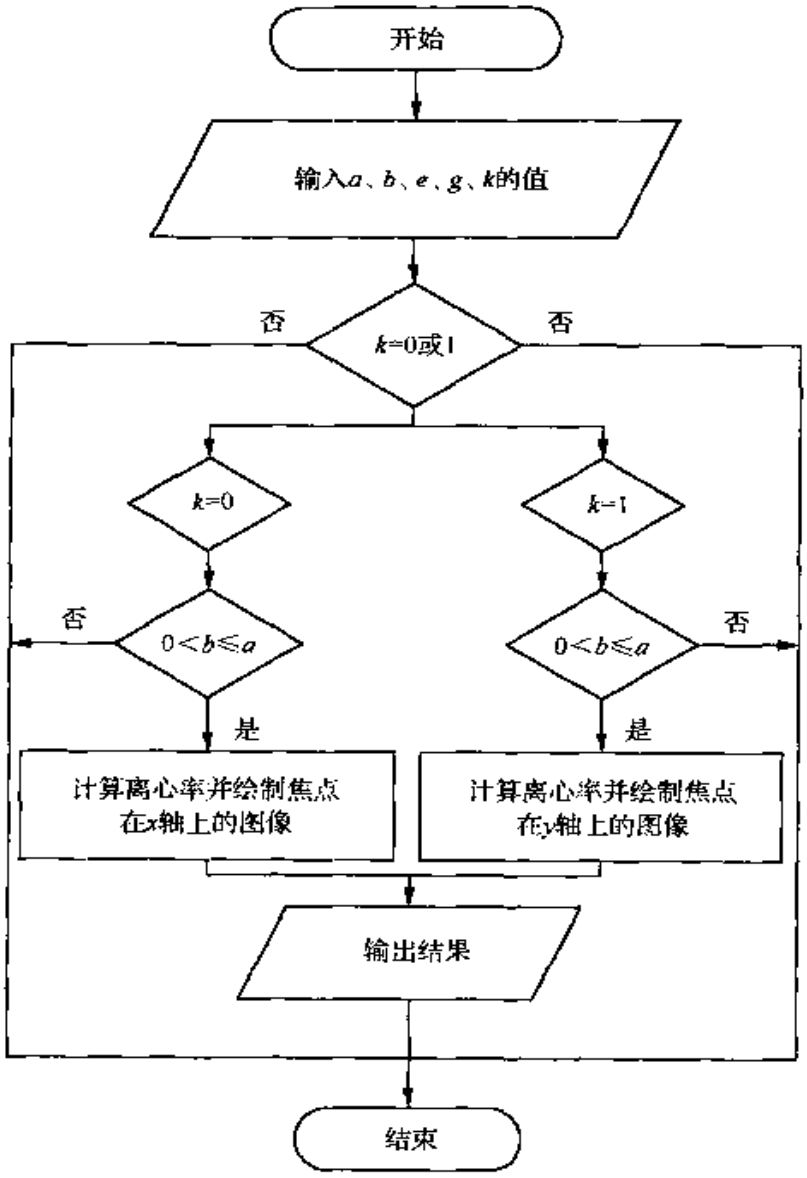


图 6.60 例(5)的执行结果

```
t = 0 : 0.01 : 2 * %pi;      //给定数组 t,确定范围及取点密度
  if b>0&b<=a
    x = a * cos(t) + e;      //椭圆的参数方程(在 x 轴上)
    y = b * sin(t) + g;
    plot2d(x,y,frameflag = 3,rect = [ - (abs(a) + abs(e) + 1), - (abs(b) + abs(g) + 1),...
        abs(a) + abs(e) + 1,abs(b) + abs(g) + 1],axesflag = 5);   //绘制椭圆的图像
    xset('color',21);          //设置坐标轴的颜色
    xset('font size',2);       //设置坐标轴的单位长度
    a1 = string(a);
    b1 = string(b);
    e1 = string(e);
```

```
    g1 = string(g);          //将 a、b、e、g 的值符号化
    xtitle('(x - ' + e1 + ')^2/' + a1 + '^2 + (y - ' + g1 + ')^2/' + b1 + '^2 = 1');  //在坐标轴上
输出椭圆的方程
  c = (a^2 - b^2)^(1/2);
  E = c/a;   //计算离心率
  m = string(E);
E = messagebox['输出离心率 e 的值:';m;],"modal", "info",['完成'])  //利用对话框输出
离心率的值
  else
  E = messagebox['Error:a、b 的值输入错误啦!!! ';],"modal", "info",['完成'])
end
  elseif k = = 1     //焦点分别在 Y 轴
  t = 0:0.01:2 * //pi;   //给定数组 t,确定范围及取点密度
  if b>0&b< = a
    y = a * cos(t) + e;   //椭圆的参数方程(在 y 轴上)
    x = b * sin(t) + g;
    plot2d(x,y,frameflag = 3,rect = [ - (abs(a) + abs(e) + 1), - (abs(b) + abs(g) + 9),...
      abs(a) + abs(e) + 1,abs(b) + abs(g) + 9],axesflag = 5);   //绘制图像
    xset('color',21);   //设置坐标轴的颜色
    xset('font size',2);   //设置坐标轴的单位长度
    a1 = string(a);
    b1 = string(b);
    e1 = string(e);
    g1 = string(g);          //将 a、b、e、g 的值符号化
    xtitle('(y - ' + e1 + ')^2/' + a1 + '^2 + (x - ' + g1 + ')^2/' + b1 + '^2 = 1');
    c = (a^2 - b^2)^(1/2);
  E = c/a;        //计算离心率
  m = string(E);
E = messagebox['输出离心率 e 的值:';m;],"modal", "info",['完成'])  //利用对话框输出
离心率的值
  else
E = messagebox['Error:a、b 的值输入错误啦  !!! ';],"modal", "info",['完成'])
  end
else
end
```

◆ **程序运行结果**

$a = 8, b = 5, e = 0, g = 0, k = 0$ 时的结果：

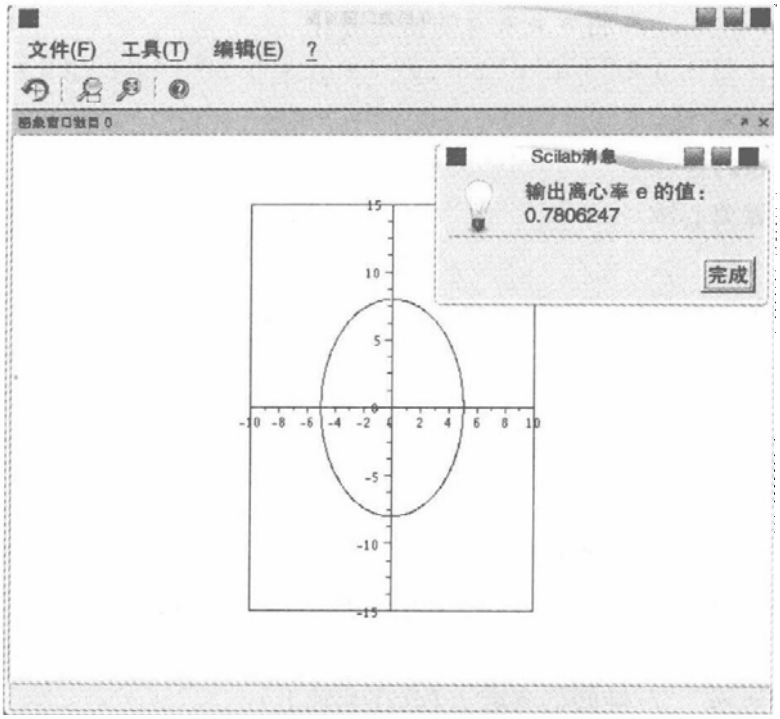


图 6.61 例(5)的执行结果

(6) 绘制一元一次函数的图像。

◆ **建模**

该算法主要用于绘制一元一次函数的图像，a、b 分别为 $y = ax + b$ 中的系数和常数项，m、n 为 x 的区间的左右端点值。

◆ **流程图**

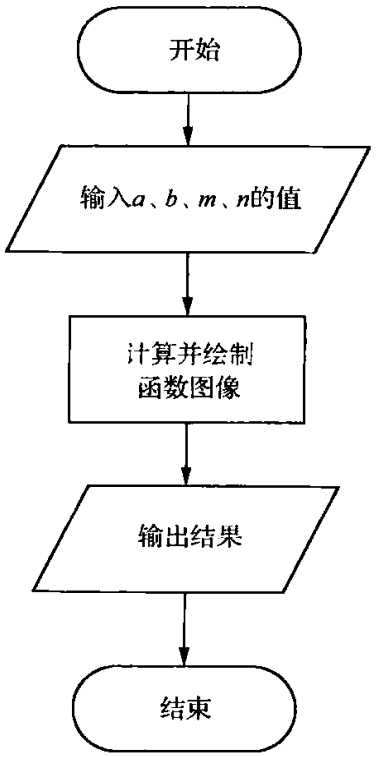


图 6.62 例(6)的流程图

◆ Scilab 程序

```
label1 = ['a :';'b :';'m :';'n :';];
B = x_mdialog(['请输入 a ,b ,m ,n 的值';'a、b 分别为 y = ax + b 中的系数和常数项';'m、n 为 x 的区间的左右端点值:'],label1,['8';'5';'-10';'10';]);
a = evstr(B(1));
b = evstr(B(2));
m = evstr(B(3));
n = evstr(B(4));      //利用对话框来输入 a、b、m、n 的值
D = messagebox(['绘制一次函数的图像?';
                '';
                '';
                '';
                ''],"modal", "info",["绘图"  "退出"]);  //询问是否继续
  x = [m : 0.01 : n];  //给定数组 x,确定范围及取点密度
  y = [a * m + b : 0.01 : a * n + b]; //给定数组 y,确定范围及取点密度
  y = a * x + b;  //计算数组 x 所对应的 y 值
  plot2d(x,y,axesflag = 5,rect = [m - 3,a * m + b - 3,n + 3,a * n + b]); //绘制图像
  xset('color',21);   //设置坐标轴的颜色
  xset('font size',2);  //设置坐标轴的单位长度
  a1 = string(a);
  b1 = string(b);   //将 a、b 的值符号化
  xtitle('y = ' + a1 + ' * x + ' + b1 + '');   //在坐标轴上输出函数表达式
```

◆ 程序运行结果

$a = 8, b = 5, m = -10, n = 10$ 时的结果：

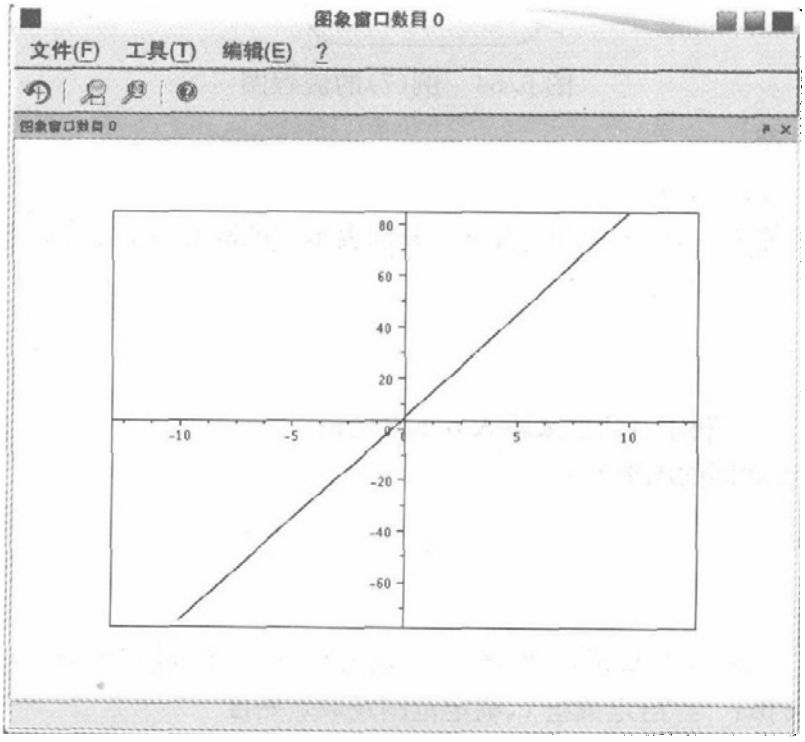


图 6.63 例(6)的执行结果

(7) 已知圆心 (a,b) 及半径 R 绘制圆的图形。

◆ **建模**

该算法主要用于在已知圆心 (a,b) 及半径 R 的情况下绘制圆的图形。

◆ **流程图**

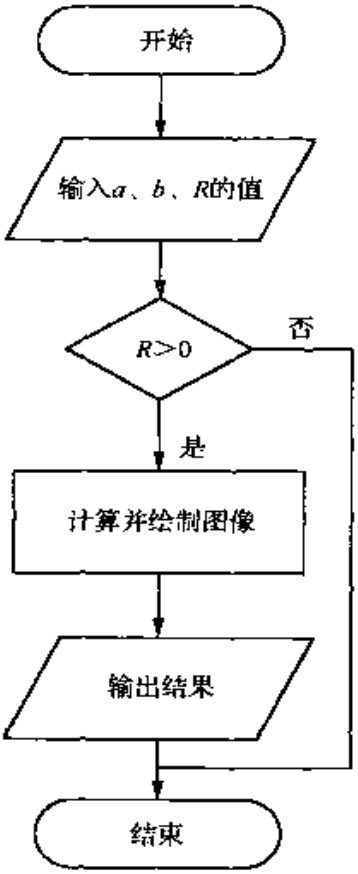


图 6.64 例(7)的流程图

◆ **Scilab 程序**

```
label1 = ['a :';'b :';'R :';];
B = x_mdialog(['请输入 a ,b ,R  的值';'a,b 分别表示圆的圆心';'R 表示半径:'],...
label1,['8';'5';'3';]);
a = evstr(B(1));
b = evstr(B(2));
R = evstr(B(3));      //利用对话框来输入 a、b、R 的值
D = messagebox(['绘制圆的图形?';
                '';
                '';
                ''],"modal", "info",["绘图"   "退出"]); //询问是否绘图
t = 0 : 0.01 : 2 * %pi;  //给定数组 t,确定范围及取点密度
if R>0      //R>0 则继续绘图,否则输出错误信息
```

```
  x = R * cos(t) + a;      //圆的参数方程
  y = R * sin(t) + b;
plot2d(x,y,frameflag = 3,rect = [ - (R + abs(a) + 1), - (R + abs(b) + 1),R + abs(a) + 1,R +
abs(b) + 1],axesflag = 5);      //绘制圆的图像
  xset('color',21);  //设置坐标轴的颜色
  r1 = string(R);
  a1 = string(a);
  b1 = string(b);   //将 a、b、R 的值符号化
  xtitle('(x - ' + a1 + ')^2 + (y - ' + b1 + ')^2 = ' + r1 + '','o(' + a1 + ' , ' + b1 + ')'); //在图像
上输出圆的方程
else
  E = messagebox['Error：半径小于  0  啦 !!! ';],"modal", "info",["完成"])
end
```

◆ **程序运行结果**

$a = 8, b = 5, R = 3$ 时的结果：

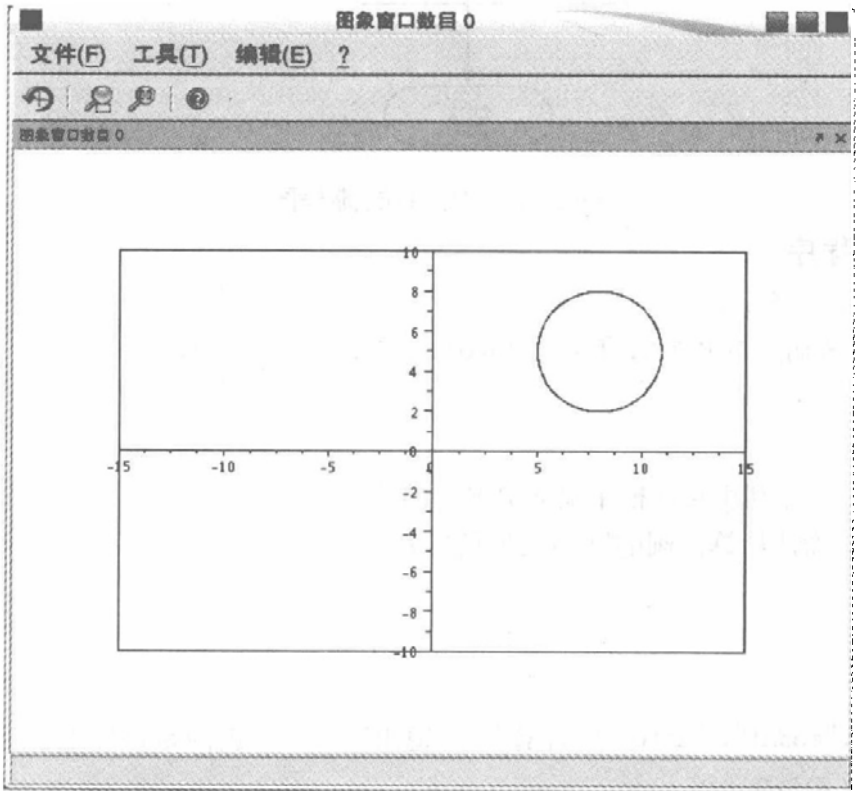


图 6.65 例(7)的执行结果

(8) 把圆的一般方程 $x^2 + y^2 + Dx + Ey + F = 0$ 标准化，并画图形。

◆ **建模**

该算法主要用于把圆的一般方程 $x^2 + y^2 + Dx + Ey + F = 0$ 标准化，并画图形。

◆ **流程图**

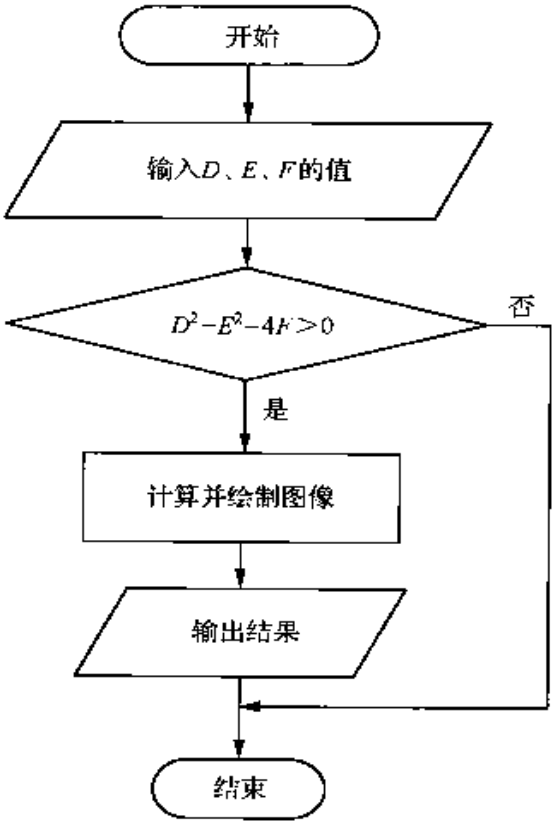


图 6.66 例(8)的流程图

◆ **Scilab 程序**

```
label1 = ['D :';'E :';'F :'];
B = x_mdialog(['请输入 D、E、F 的值 :'],label1,['2';'4';'-6';]);
D = evstr(B(1));
E = evstr(B(2));
F = evstr(B(3));    //利用对话框来输入 D、E、F 的值
D = messagebox(['确认计算该圆的圆心及半径?';
                '';
                '';
                ''],"modal", "info",["计算"    "退出"]);  //询问是否计算
if (D^2 + E^2 - 4 * F)>0
  a = -D/2;
  b = -E/2;    //a、b 分别表示圆心的横、纵坐标
  r = 1/2 * (sqrt(D^2 + E^2 - 4 * F));  //r 为圆的半径
  c = string(a);
  d = string(b);
  e = string(r);    //将 a、b、r 的值符号化
```

```
E = messagebox(['输出圆的圆心（a，b）及半径 R 的值:';c;d;e;],"modal", "info",["绘图"  "退出"])  //利用对话框输出圆心及半径的值
t = 0 : 0.01 : 2 * %pi;  //t 为弧角,给定数组 t,确定范围及取点密度
    x = r * cos(t) + a;
    y = r * sin(t) + b;  //圆的参数方程
plot2d(x,y,frameflag = 3,rect = [ - (r + abs(a) + 3), - (r + abs(b) + 1),(r + abs(a) + 3),(r + abs(b) + 1)],axesflag = 5);  //绘制圆的图像
    xset('color',21);      //设置坐标轴的颜色
    xset('font size',2);      //设置坐标轴的单位长度
    a1 = string(a);
    b1 = string(b);
    r1 = string(r);  //将 a、b、r 的值符号化
    xtitle('(x - ' + a1 + ')^2 + (y - ' + b1 + ')^2 = ' + r1 + '','o(' + a1 + ' , ' + b1 + ')'); //在坐标轴上输出表达式
E = messagebox(['输出圆的圆心（a，b）及半径 R 的值:';c;d;e;],"modal", "info",["完成"])
    //显示完图像后再次利用对话框输出圆心及半径的值
else
      E = messagebox(['输出 Error:D^2 + E^2 - 4 * F 小于 0  啦!!! ';],"modal", "info",["完成"])
end
```

◆ **程序运行结果**

$D=2, E=4, F=-6$ 时的结果：

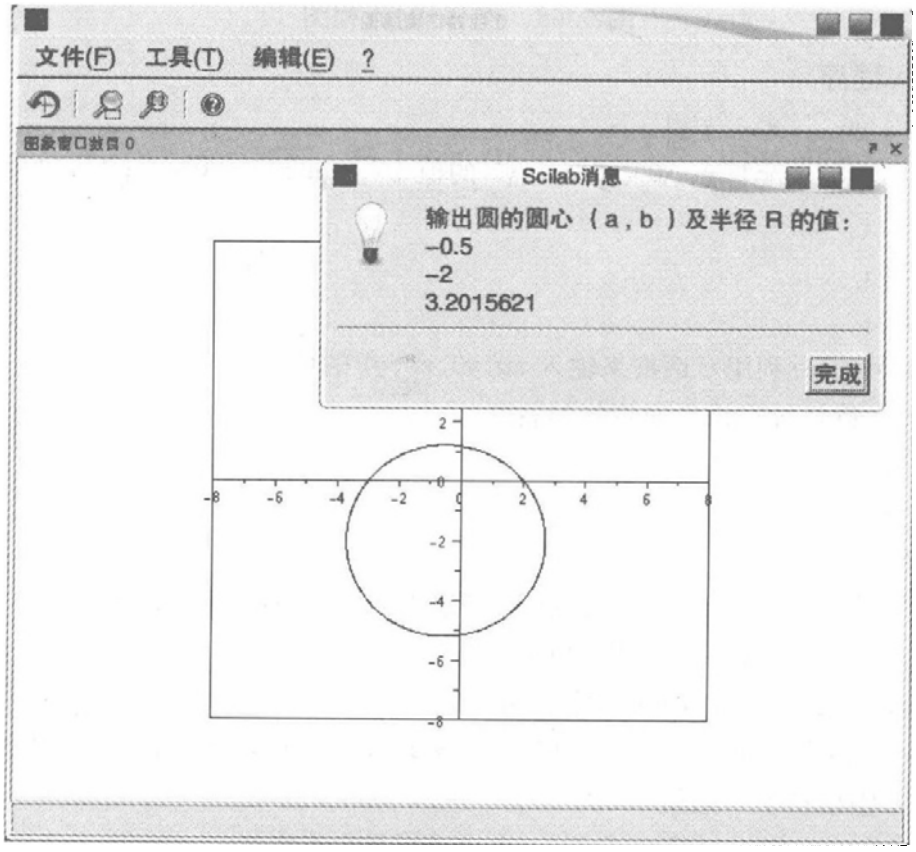


图 6.67 例(8)的执行结果

(9) 已知两点的坐标,计算过这两点的直线的斜率并绘制该直线的图像。

◆ **建模**

该算法主要用于已知两点的坐标,计算过这两点的直线的斜率并绘制该直线的图像。x0 为第一点的横坐标,y0 为第一点的纵坐标,x1 为第二点的横坐标,y1 为第二点的纵坐标。

◆ **流程图**

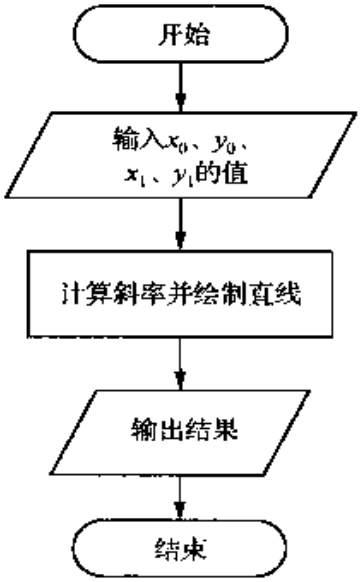


图 6.68 例(9)的流程图

◆ **Scilab 程序**

```
label1 = ['x0 :';'y0 :';'x1 :';'y1';];
B = x_mdialog(['请输入坐标点 x0 、y0 、x1、y1 的值 :'],label1,['0';'0';'1';'3';]);
x0 = evstr(B(1));
y0 = evstr(B(2));
x1 = evstr(B(3));
y1 = evstr(B(4));    //利用对话框来输入 x0、y0、x1、y1 的值
D = messagebox(['确认计算该平面直线的斜率?';
                '';
                '';
                ''],"modal", "info",["计算"    "退出"]); //询问是否计算
    k = (y1 - y0)/(x1 - x0);   //计算斜率
    k1 = string(k);    //将斜率的值符号化
E = messagebox(['该直线的斜率 k 为:';k1;],"modal", "info",["绘图"      "退出"]);
//利用对话框来输出斜率 k 的值
    x = - 10 : 0.01 : 10;   //给定数组 x,确定范围及取点密度
    y = k * x + y1 - k * x1;   //计算出对应的数组 y
```

```
plot2d(x,y,axesflag = 5);  //绘制直线
    xset('color',21);     //设置坐标轴的颜色
    xset('font size',2);  //设置坐标轴的单位长度
    k1 = string(k);
    xtitle('y = k * x + y1 - k * x1','k = ' + k1 + '');   //在图像上输出直线的方程
E = messagebox['该直线的斜率 k 为:';k1;],"modal", "info",['完成'] //再次利用对话框来输出斜率 k 的值
```

◆ **程序运行结果**

$(x_0, y_0) = (0,0), (x_1, y_1) = (1,3)$ 时的结果：

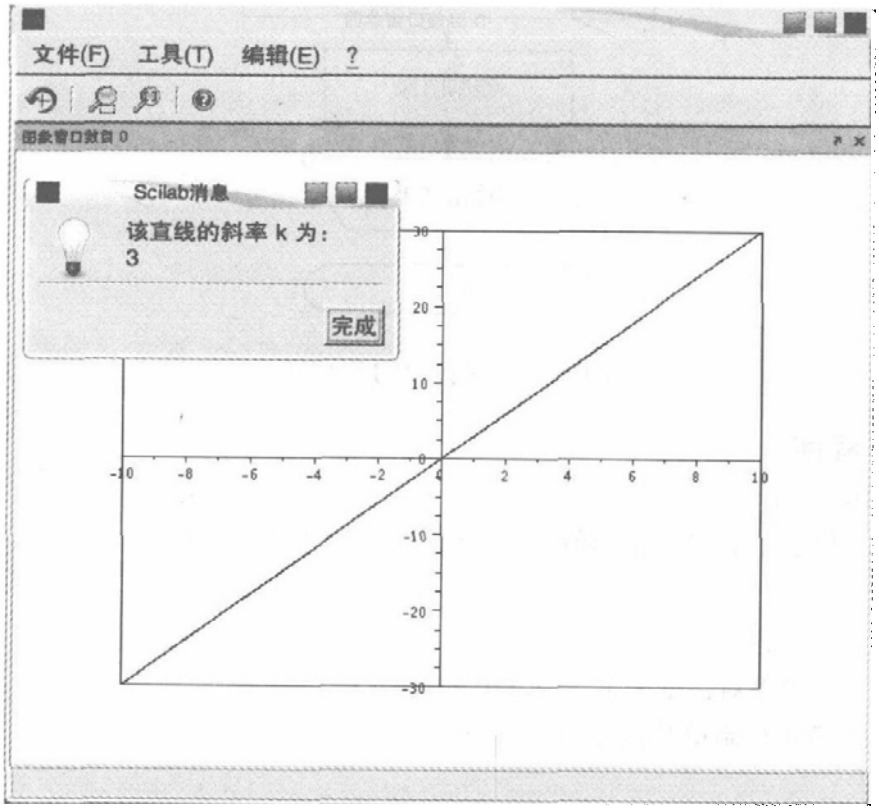


图 6.69 例(9)的执行结果

6.8 空间解析几何的作图

(1) 绘制单叶双曲面图像。

◆ **建模**

该算法主要用于绘制单叶双曲面 $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$ 的图像。其中 a、b、c 分别为 $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$ 中的系数。

◆ 流程图

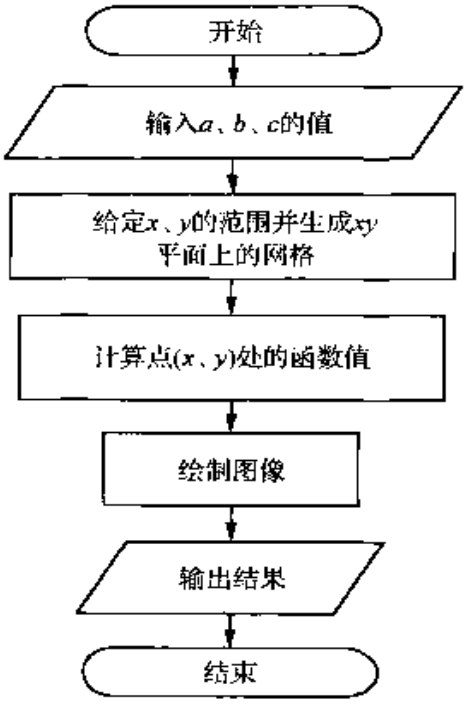


图 6.70 例(1)的流程图

◆ Scilab 程序

```
label1 = ['a:';'b:';'c';];
B = x_mdialog['请输入 a , b , c 的值 : '],label1,['1';'2';'3';]);
a = evstr(B(1));
b = evstr(B(2));
c = evstr(B(3)); //利用对话框来输入 a,b,c 的值
D = messagebox(['确认绘制单叶双曲面? ';
                '';
                '';
                ''],"modal", "info",['绘制 ','退出 ']);  //询问是否继续绘图
[X,Y] = meshgrid( - 1 : 0.02 : 1, - 1 : 0.02 : 1); //确定 x,y 的范围并生成 xy 平面上的网格
Z = ((X.^2/(a^2) + Y.^2/(b^2) - 1) * c^2)^(1/2);   //计算在点(X,Y)处的函数值
Z = - ((X.^2/(a^2) + Y.^2/(b^2) - 1) * c^2)^(1/2);
surf(X,Y,Z);  //绘制图像
```

◆ 程序运行结果

$a = 1, b = 2, c = 3$ 时的结果：

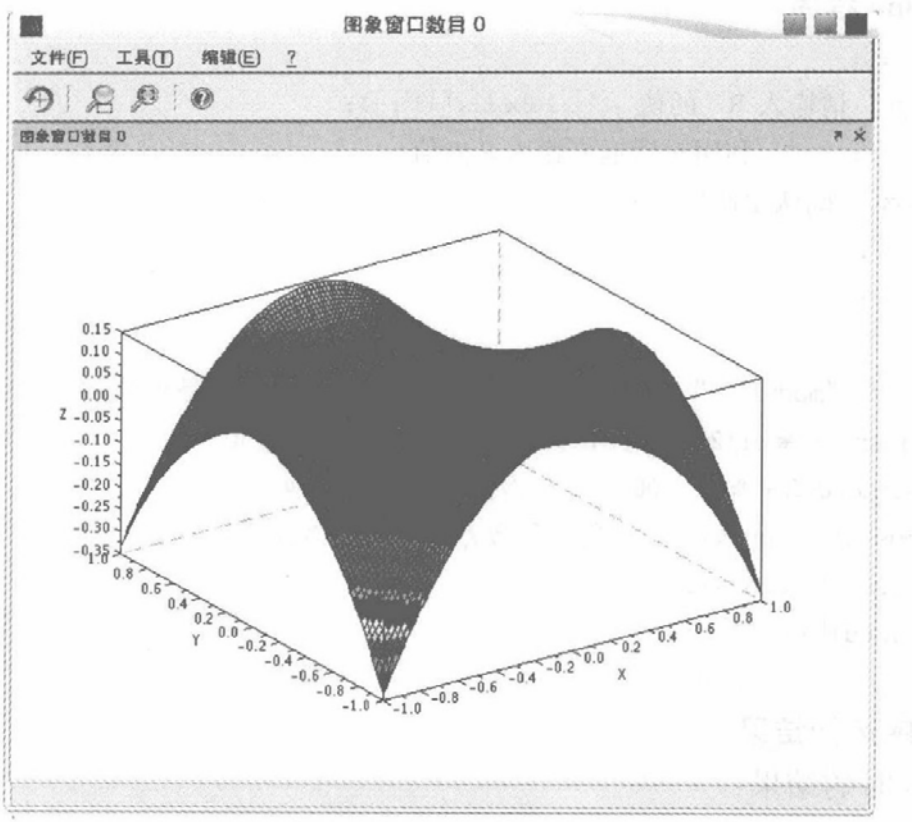


图 6.71　例(1)的执行结果

(2) 绘制球的图像。

◆ **建模**

该算法主要用于绘制球 $x^2+y^2+z^2=R^2$ 的图像。

◆ **流程图**

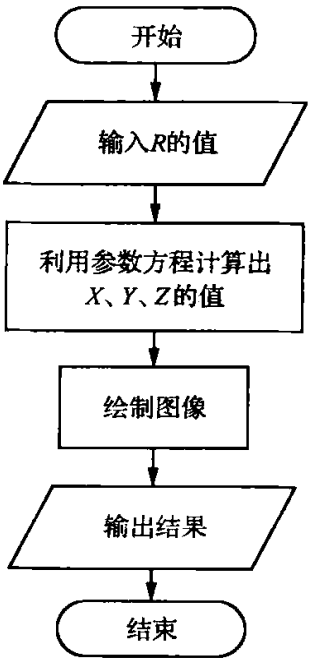


图 6.72　例(2)的流程图

◆ **Scilab 程序**

```
label1 = ['R:';];
B = x_mdialog(['请输入 R  的值 :'],label1,['3';]);
R = evstr(B(1));     //利用对话框来输入 R 的值
D = messagebox(['确认绘制球?';
                '';
                '';
                '';
                ''],"modal", "info",['绘制','退出']);  //询问是否绘制图像
  u = linspace( - %pi/2, %pi/2,100);  //给定线性数组 u
  v = linspace(0,2 * %pi,100);      //给定线性数组 v
  X = R * cos(u)' * cos(v);   //利用参数方程计算出 X、Y、Z
  Y = R * cos(u)' * sin(v);
  Z = R * sin(u)' * ones(v);
  surf(X,Y,Z);  //绘制图像
```

◆ **程序运行结果**

$R = 3$ 时的结果：

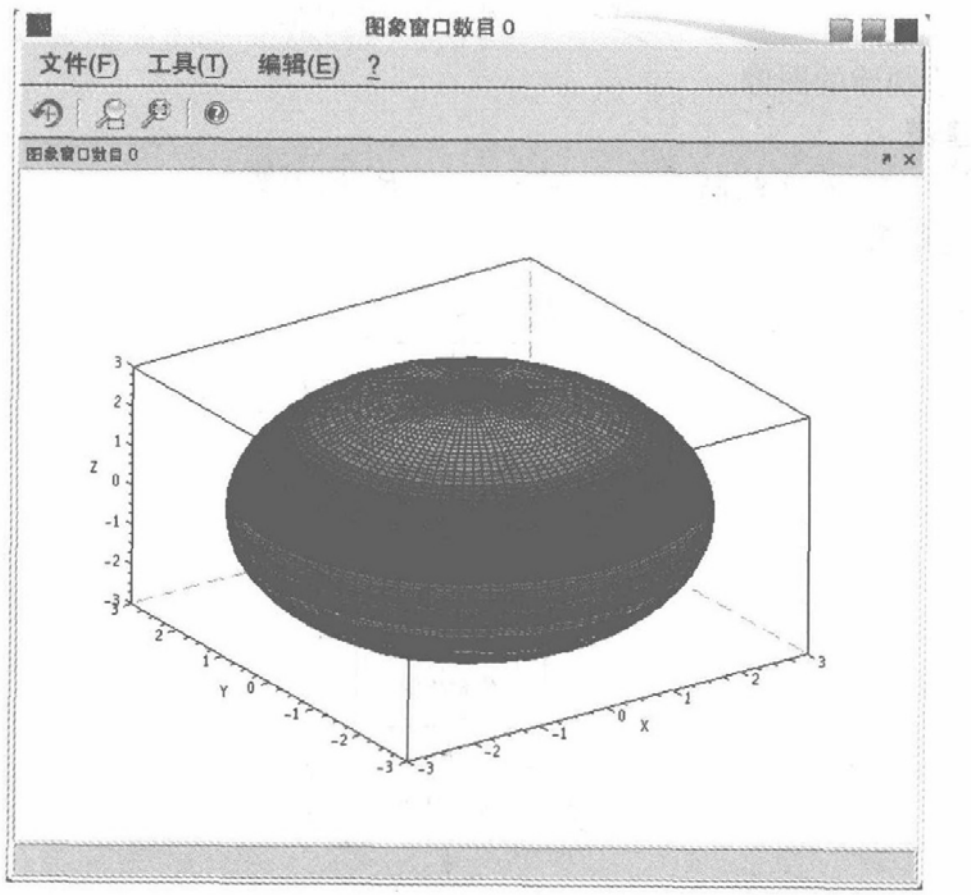


图 6.73　例(2)的执行结果

(3) 绘制双曲抛物面的图像。

◆ **建模**

该算法主要用于绘制双曲抛物面 $\frac{x^2}{a^2}-\frac{y^2}{b^2}=2z$ 的图像。

◆ **流程图**

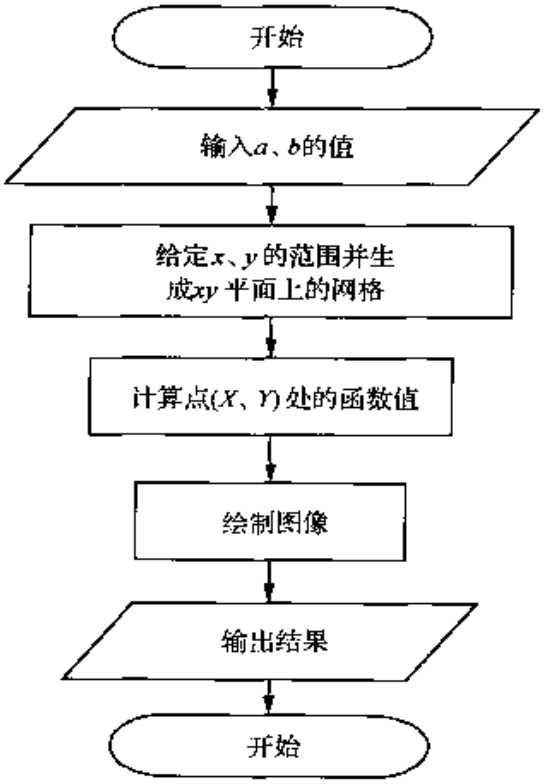


图 6.74　例(3)的流程图

◆ **Scilab 程序**

```
label1 = ['a:';'b:';];
B = x_mdialog(['请输入 a , b  的值 : '],label1,['1';'2';]);
a = evstr(B(1));
b = evstr(B(2));  //利用对话框来输入 a , b 的值
D = messagebox(['确认绘制双曲抛物面? ';
                '';
                '';
                ''],"modal", "info",['绘制 ','退出 ']);  //询问是否绘图
[X,Y] = meshgrid(-1:0.02:1,-1:0.02:1);  //确定 X、Y 的范围并生成 XY 平面上的网格
  Z = X.^2/(2*a^2) - Y.^2/(2*b^2);  //计算在点(X,Y)处的函数值
  xtitle('X.^2/a^2 - Y.^2/b^2 = 2 * z');  //在图形上标注双曲抛物面的表达式
  surf(X,Y,Z);  //绘制图像
```

◆ **程序运行结果**

$a = 1, b = 2$ 时的结果：

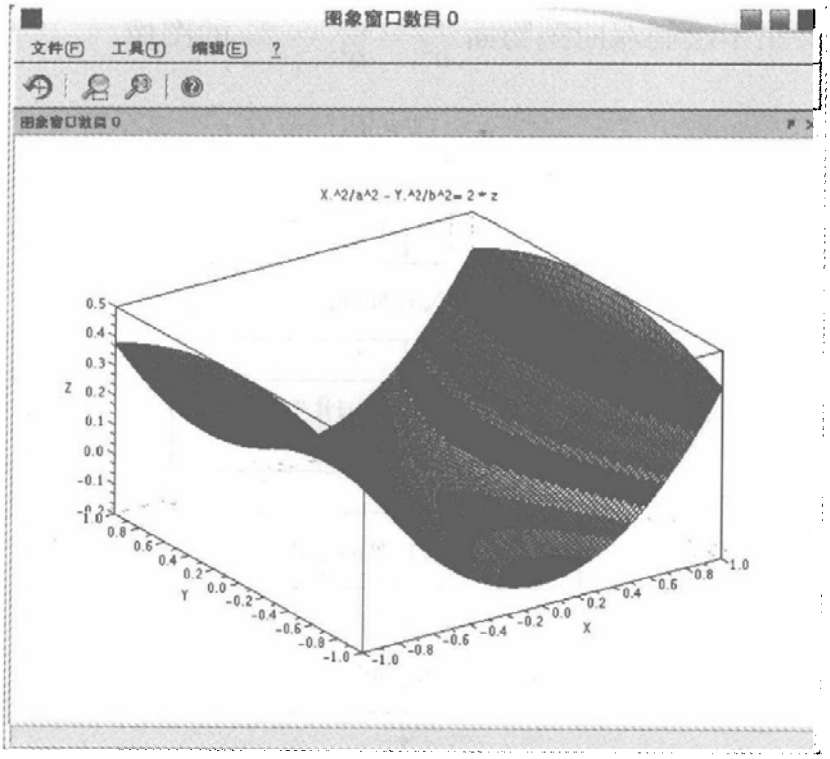


图 6.75 例(3)的执行结果

(4) 绘制双叶双曲面的图像。

◆ **建模**

该算法主要用于绘制双叶双曲面 $\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=-1$ 的图像。

◆ **流程图**

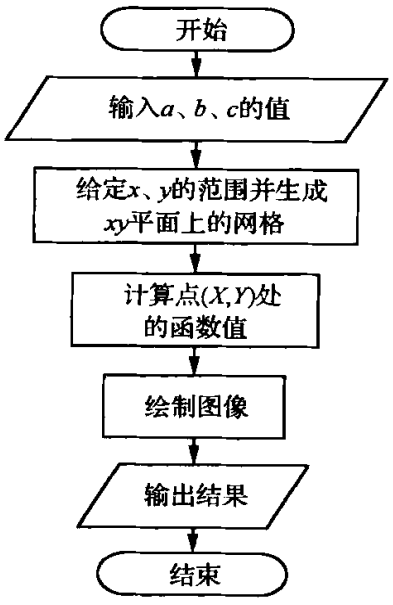


图 6.76 例(4)的流程图

◆ **Scilab 程序**

```
label1 = ['a:';'b:';'c';];
B = x_mdialog(['请输入 a，b，c 的值：'],label1,['1';'2';'3';]);
a = evstr(B(1));
b = evstr(B(2));
c = evstr(B(3));  //利用对话框来输入 a，b，c 的值
D = messagebox(['确认绘制双叶双曲面?';
                '';
                '';
                ''],"modal", "info",['绘制','退出']);  //询问是否继续绘图
[X,Y] = meshgrid( - 2 : 0.02 : 2, - 2 : 0.02 : 2);  //确定 x、y 的范围并生成 xy 平面上的网格
  Z = ((X.^2/(a^2) + Y.^2/(b^2) + 1) * c^2)^(1/2);  //计算在点(X,Y)处的函数值
  surf(X,Y,Z); //绘制图像
  Z = - ((X.^2/(a^2) + Y.^2/(b^2) + 1) * c^2)^(1/2);  //计算在点(X,Y)处的函数值
  surf(X,Y,Z);  //绘制图像
```

◆ **程序运行结果**

$a = 1, b = 2, c = 3$ 时的结果：

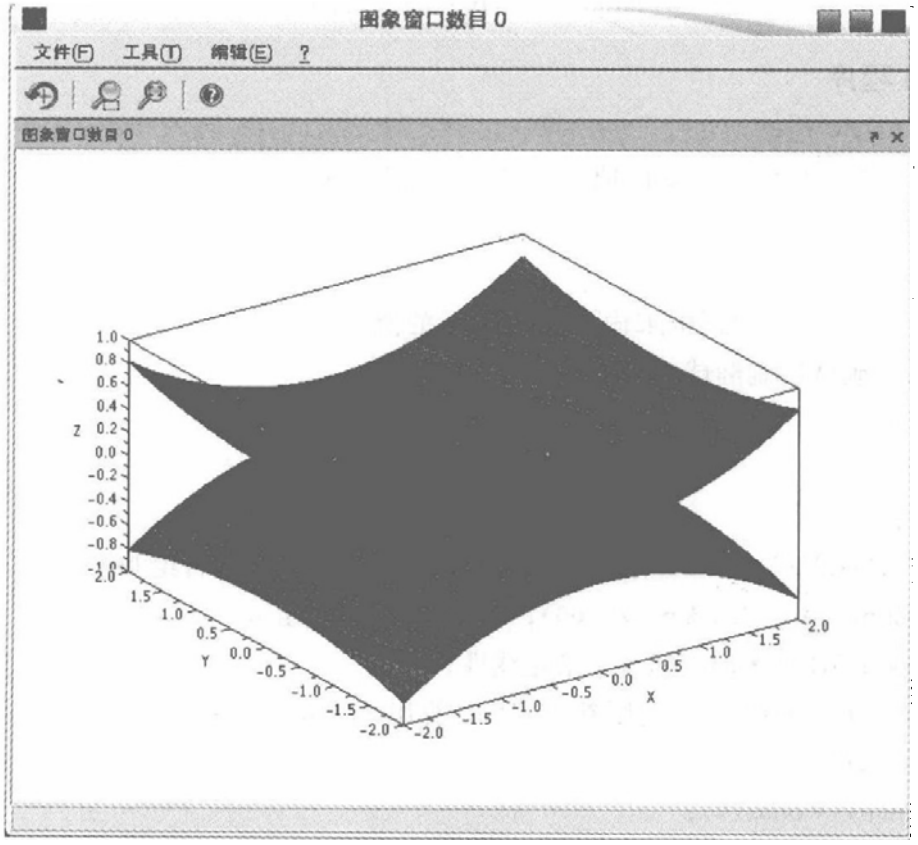


图 6.77　例(4)的执行结果

(5) 绘制椭球的图像。

◆ **建模**

该算法主要用于绘制椭球 $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$ 的图像。

◆ **流程图**

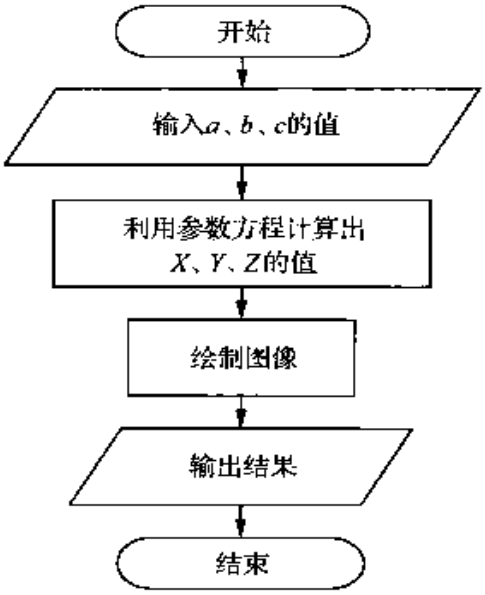


图 6.78 例(5)的流程图

◆ **Scilab 程序**

```
label1 = ['a:';'b:';'c';];
B = x_mdialog(['请输入 a , b , c 的值 : '],label1,['1';'2';'3';]);
a = evstr(B(1));
b = evstr(B(2));
c = evstr(B(3));  //利用对话框来输入 a , b , c 的值
D = messagebox(['确认绘制椭球? ';
                '';
                '';
                ''],"modal", "info",['绘制 ','退出 ']);  //询问是否绘制图形
   u = linspace( - %pi/2, %pi/2,100);  //给定线性数组 u
   v = linspace(0,2 * %pi,100);//给定线性数组 v
   X = a * cos(u)' * cos(v);//利用参数方程计算出 X、Y、Z
   Y = b * cos(u)' * sin(v);
   Z = c * sin(u)' * ones(v);
 plot3d3(X,Y,Z); //绘制图形
```

◆ **程序运行结果**

$a = 1, b = 2, c = 3$ 时的结果：

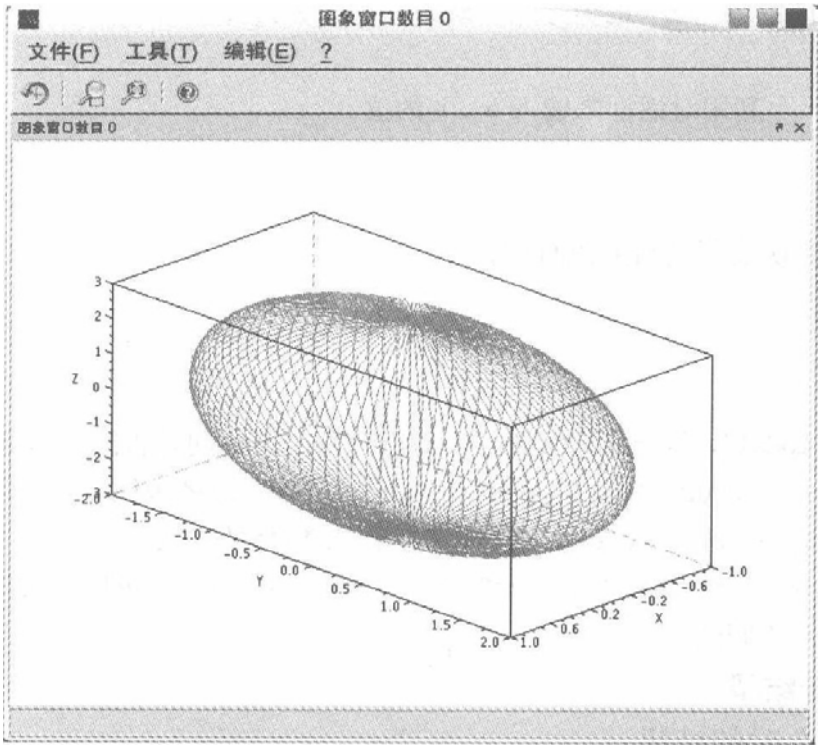


图 6.79　例(5)的执行结果

(6) 绘制椭圆抛物面的图像。

◆ **建模**

该算法主要用于绘制椭圆抛物面 $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 2z$ 的图像。

◆ **流程图**

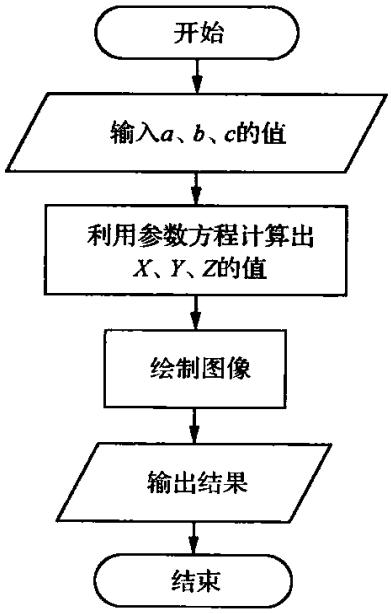


图 6.80　例(6)的流程图

◆ **Scilab 程序**

```
label1 = ['a:';'b:';];
B = x_mdialog(['请输入 a，b 的值：'],label1,['1';'2';]);
a = evstr(B(1));
b = evstr(B(2));   //利用对话框来输入 a，b 的值


D = messagebox(['确认绘制椭圆抛物面?';
                '';
                '';
                '';
                ''],"modal", "info",['绘制','退出']);  //询问是否绘制图形
[X,Y] = meshgrid(-1:0.02:1,-1:0.02:1);  //确定 x、y 的范围并生成 xy 平面上的网格
  Z = X.^2/(2*a^2) + Y.^2/(2*b^2);   //计算在点(X,Y)处的函数值
  xtitle('Z = X.^2/(2*a^2) + Y.^2/(2*b^2)');  //在图形上标注椭圆抛物面的表达式
  surf(X,Y,Z);  //绘制图形
```

◆ **程序运行结果**

$a = 1, b = 2$ 时的结果：

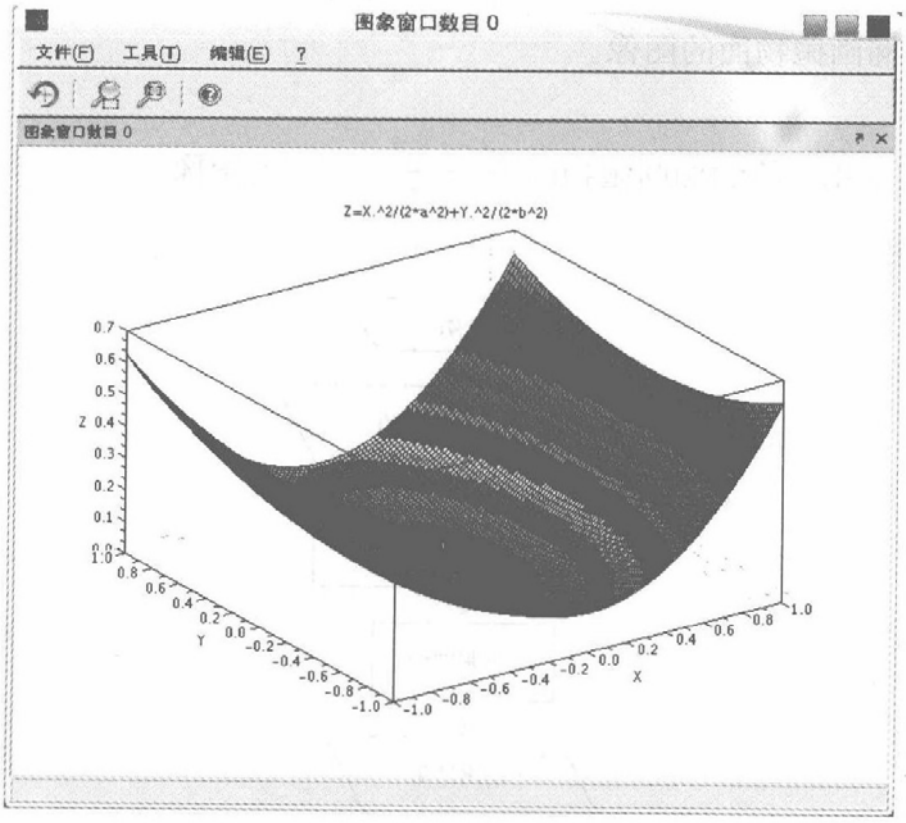


图 6.81 例(6)的执行结果

参 考 文 献

胡包钢，赵星，康孟珍．2003. 科学计算自由软件——Scilab 教程．北京：清华大学出版社

黄铎，王风，李志伟．2006. 科学计算自由软件 Scilab 基础教程．北京：清华大学出版社

刘强．2005. 信息技术与数学课程整合的理论及实践．山东：山东师范大学

吕林根，许子道．2006. 解析几何．北京：高等教育出版社

吴华洋．2008. 基于 Linux 环境的计算机基础教程(第 2 版)．北京：清华大学出版社

中华人民共和国教育部．2003. 普通高中数学课程标准(实验)．北京：人民教育出版社

周开利，邓春晖．2007. MATLAB 基础及其应用教程．北京：北京大学出版社

http://course 2007. szu. edu. cn/jisuanji/courseware/siExperiment/2008experiment12. htm

http://hkumath. hku. hk/~nkt/Scilab/IntroToScilab. html

Scilab 中文教程 v0. 04. http://www. scilab. org. cn

相 关 网 站

Scilab 主页	http://www. scilab. org
Scilab 中文网站	http://www. scilab. org. cn
基础教育科学计算教学服务社区	http://scilab. gznu. edu. cn

附录　Scilab 常用命令与函数

1. 通用指令

help 在线帮助
apropos 文档中关键词搜寻
ans 缺省变量名以及最新表达式的运算结果
clear 从内存中清除变量和函数
exit 关闭 Scilab
quit 退出 Scilab
save 把内存变量存入磁盘
exec 运行脚本文件
mode 文件运行中的显示格式
getversion 显示 Scilab 版本
ieee 浮点运算溢出显示模式选择
who 列出工作内存中的变量名
edit 文件编辑器
type 变量类型
what 列出 Scilab 基本命令
format 设置数据输出格式
chdir 改变当前工作目录
getenv 给出环境值
mkdir 创建目录
pwd 显示当前工作目录
evstr 执行表达式

2. 运算符和特殊算符

＋ 加
－ 减
＊ 矩阵乘
.＊ 数组乘
^ 矩阵乘方
.^ 数组乘方
\ 反斜杠或左除
/ 斜杠或右除
./或 . 数组除
== 等号
～= 不等号
＜ 小于
＞ 大于
＝ 大于或等于
&，and 逻辑与
|，or 逻辑或
～，not 逻辑非
：冒号
（）圆括号
[ ] 方括号
{ } 花括号
. 小数点
，逗号
；分号
// 注释号
＝ 赋值符号
' 引号
' 复数转置号
.' 转置号
ans 最新表达式的运算结果
%eps 浮点误差容限
%i 虚数单位 ＝ $\sqrt{(-1)}$
%inf 正无穷大

%pi 圆周率，π=3.1415926535897....

3. 编程语言结构

abort 中止计算或循环
break 终止最内循环
case 同 select 一起使用
continue 将控制转交给外层的 for 或 while 循环
else 同 if 一起使用
elseif 同 if 一起使用
end 结束 for,while,if 语句
for 按规定次数重复执行语句
if 条件执行语句
otherwise 可同 switch 一起使用
pause 暂停模式
return 返回
select 多个条件分支
then 同 if 一起使用
while 不确定次数重复执行语句
eval 特定值计算
feval 函数特定值计算或多变量计算
function 函数文件头
global 定义全局变量
isglobal 检测变量是否为全局变量
error 显示错误信息
lasterror 显示最近的错误信息
sprintf 按格式把数字转换为串
warning 显示警告信息

4. 基本数学函数

acos 反余弦
acosh 反双曲余弦
acot 反余切
acoth 反双曲余切
acsc 反余割
acsch 反双曲余割
asin 反正弦
asinh 反双曲正弦
atan 反正切
atanh 反双曲正切
cos 余弦
cosh 双曲余弦
cotg 余切
coth 双曲余切
sin 正弦
sinh 双曲正弦
tan 正切
tanh 双曲正切
exp 指数
log 自然对数
log10 常用对数
log2 以 2 为底的对数
sqrt 平方根
abs 绝对值
conj 复数共轭
imag 复数虚部
real 复数实部
ceil 向上(正无穷大方向)取整
fix 向零方向取整
floor 向下(负无穷大方向)取整
round 四舍五入取整
sign 符号函数
gsort 降次排序
erf 误差函数
erfc 补误差函数
gamma gamma 函数
interp 插值函数
interpln 线性插值函数
intsplin 样条插值函数
smooth 样条平滑函数

spline 样条函数
quarewave 方波函数
sign 符号函数
double 将整数转换为双精度浮点数

5. 基本矩阵函数和操作

eye 单位阵
zeros 全零矩阵
ones 全 1 矩阵
rand 均匀分布随机阵
genmarkov 生成随机 Markov 矩阵
linspace 线性等分向量
logspace 对数等分向量
logm 矩阵对数运算
cumprod 矩阵元素累计乘
cumsum 矩阵元素累计和
toeplitz Toeplitz 矩阵
disp 显示矩阵和文字内容
length 确定向量的长度
size 确定矩阵的维数
diag 创建对角阵或抽取对角向量
find 找出非零元素 1 的下标
matrix 矩阵变维
rot90 矩阵逆时针旋转 90 度
sub2ind 据全下标换算出单下标
tril 抽取下三角阵
triu 抽取上三角阵
conj 共轭矩阵
companion 伴随矩阵
det 行列式的值
norm 矩阵或向量范数
nnz 矩阵中非零元素个数
null 清空向量或矩阵中的某个元素
orth 正交基
rank 矩阵秩
trace 矩阵迹
cond 矩阵条件数
rcond 逆矩阵条件数
inv 矩阵的逆
lu LU 分解或高斯消元法
pinv 伪逆
qr QR 分解
givens Givens 变换
linsolve 求解线性方程
lyap Lyapunov 方程
hess Hessenberg 矩阵
poly 特征多项式
schur Schur 分解
expm 矩阵指数
expm1 矩阵指数的 Pade 逼近
expm2 用泰勒级数求矩阵指数
expm3 通过特征值和特征向量求矩阵指数
funm 计算一般矩阵函数
logm 矩阵对数
sqrtm 矩阵平方根

6. 特性值与奇异值

spec 矩阵特征值
gspec 矩阵束特征值
bdiag 块矩阵，广义特征向量
eigenmarkov 正则化 Markov 特征向量
pbig 特征空间投影
svd 奇异值分解
sva 奇异值分解近似

7. 矩阵元素运算

cumprod 元素累计积
cumsum 元素累计和

hist 统计频数直方图
max 最大值
mean 平均值
median 中值
min 最小值
prod 元素积
sort 由大到小排序
std 标准差
sum 元素和
trapz 梯形数值积分
corr 求相关系数或方差

8. 稀疏矩阵运算

sparse 稀疏矩阵(只存储非零元素)
adj2sp 邻接矩阵转换为稀疏矩阵
full 稀疏矩阵转换为全矩阵
mtlb_sparse 将 Scilab 稀疏矩阵转换为 MATLAB 稀疏矩阵格式
sp2adj 稀疏矩阵转换为邻接矩阵
speye 稀疏矩阵方式单位阵
sprand 稀疏矩阵方式随机矩阵
spzeros 稀疏矩阵方式全零阵
lufact 稀疏矩阵 LU 分解
lusolve 稀疏矩阵方程求解
spchol 稀疏矩阵 Cholesky 分解

9. 输入输出函数

diary 生成屏幕文本记录
disp 变量显示
file 文件管理
input 用户键盘输入
load 读已存的变量
mclose 关闭文件
mget 读二进制文件
mgetl 按行读 ASCII 码文件
mgetstr 读字符串中单个字
mopen 打开文件
mput 写二进制文件
mfscanf 读 ASCII 码文件
print 将变量记录为文件
read 读矩阵变量
save 存变量为二进制文件
strartup 启动文件
write 按格式存文件
xgetfile 对话方式获取文件路径
x_dialog 建立 Xwindow 参数输入对话框
Tk_Getvar 得到 Tk 文件变量
Tk_EvalFile 执行 Tk 文件

10. 函数与函数库操作

deff 在线定义函数
edit 函数编辑器
function 打开函数定义
functions Scilab 函数或对象
genlib 在给定目录下建立所有文件的函数库
get_function_path 读函数库的文件存储目录路径
getd 读函数库中的全部文件
getf 在文件中定义一个函数
lib 函数库定义
macro Scilab 函数或对象
macrovar 输入变量个数
newfun 输出变量个数

11. 字符串操作

code2str 将 Scilab 数码转换为字符串
convstr 字母大小转换
emptystr 清空字符串
grep 搜寻相同字符串

part 字符提取
str2code 将字符串转换为 Scilab 数码
string 字符串转换
strings Scilab 对象，字符串
strcat 连接字符
strindex 字符串的字符位置搜寻
strsubst 字符串中的字符替换

12. 日期与时间

date 日期
getdate 读日期与时间
timer CPU 时间计时

13. 二维图形函数

plot2d 直角坐标下线性刻度曲线
champ 2 维向量场
champ1 由颜色箭头表示的 2 维向量场
contour2d 等高线图
errbar 曲线上增加误差范围框线条
grayplot 应用颜色表示的表面
xgrid 画坐标网格线
histplot 统计频数直方图
Matplot 散点图阵列

14. 三维图形函数

plot3d 三维表面
plot3d1 用颜色或灰度表示的三维表面
param3d 三维中单曲线
param3d1 三维中多曲线
contour 三维表面上的等高线图
hist3d 三维表示的统计频数直方图
geom3d 三维向二维上的投影

15. 线条类图形

xpoly 单线条或单多边形
xpolys 多线条或多各多边形
xrpoly 正多边形
xsegs 非连接线段
xfpoly 单个多边形内填充
xfpolys 多个多边形内填充
xrect 矩形
xfrect 单个矩形内填充
xrects 多个矩形内填充
xarc 单个弧线段或弧园
xarcs 多个弧线段或弧园
xfarc 单个弧线段或弧园填充
xfarcs 多个弧线段或弧园填充
xarrows 多箭头

16. 图形注释，变换

xstring 图形中字符
xstringb 框内字符
xtitle 图形标题
xaxis 轴名标注
plotframe 图形加框并画坐标网格线
isoview 等尺寸比例显示(原图形窗口不改变)
square 等尺寸比例显示(原图形窗口改变)
xsetech 设置小窗口
xchange 转换实数为图形像素坐标值
subplot 设置多个子窗口

17. 图形颜色及图形文字

colormap 应用颜色图
getcolor 交互式选择颜色图
addcolor 增加新色于颜色图
graycolormap 线性灰度图
hotcolormap 热色(红到黄色)颜色图
xset 图形显示方式设定

xget 读当前图形显示方式设定
getsymbol 交互式选择符号和尺寸

18. 图形文件及图形文字

xsave 将图形存储为文件
xload 从磁盘中读出图形文件
xbasimp 将图形按 PS 文件打印或存储为文件
xs2fig 将图形生成 Xfig 格式文件
xbasc 取消图形窗及其相关内容
xclear 清空图形窗
driver 选择图形驱动器
xinit 图形驱动器初始化
xend 关闭图形
xbasr 图形刷新
replot 更改显示范围后的图形刷新
xdel 关闭图形
xname 改变当前图形窗名称

19. 多项式计算

coeff 多项式系数
coffg 多项式矩阵逆
degree 多项式阶数
denom 分母项
derivat 有理矩阵求导
determ 矩阵行列式值
factors 因式分解
hermit Hermit 型
horner 多项式计算
invr 有理矩阵逆
lcm 最小公倍数
ldiv 多项式矩阵长除
numer 分子项
pdiv 多项式矩阵除
pol2des 多项式矩阵到表达式变换
pol2str 多项式到字符串变换
polfact 最小因式
residu 余量
roots 多项式根
simp 多项式化简
systmat 系统矩阵